

UUU	UUU	EEEEEEEEEEEEEEEE	TTTTTTTTTTTTTTTT	PPPPPPPPPPPPPP	
UUU	UUU	EEEEEEEEEEEEEEEE	TTTTTTTTTTTTTTTT	PPPPPPPPPPPPPP	
UUU	UUU	EEEEEEEEEEEEEEEE	TTTTTTTTTTTTTTTT	PPPPPPPPPPPPPP	
UUU	UUU	EEE	TTT	PPP	PPP
UUU	UUU	EEE	TTT	PPP	PPP
UUU	UUU	EEE	TTT	PPP	PPP
UUU	UUU	EEE	TTT	PPP	PPP
UUU	UUU	EEE	TTT	PPP	PPP
UUU	UUU	EEE	TTT	PPP	PPP
UUU	UUU	EEE	TTT	PPP	PPP
UUU	UUU	EEEEEEEEEEEEEEEE	TTT	PPPPPPPPPPPPPP	
UUU	UUU	EEEEEEEEEEEEEEEE	TTT	PPPPPPPPPPPPPP	
UUU	UUU	EEEEEEEEEEEEEEEE	TTT	PPPPPPPPPPPPPP	
UUU	UUU	EEE	TTT	PPP	
UUU	UUU	EEE	TTT	PPP	
UUU	UUU	EEE	TTT	PPP	
UUU	UUU	EEE	TTT	PPP	
UUU	UUU	EEE	TTT	PPP	
UUU	UUU	EEE	TTT	PPP	
UUU	UUU	EEE	TTT	PPP	
UUUUUUUUUUUUUUUU	UUUUUUUUUUUUUUUU	EEEEEEEEEEEEEEEE	TTT	PPP	
UUUUUUUUUUUUUUUU	UUUUUUUUUUUUUUUU	EEEEEEEEEEEEEEEE	TTT	PPP	
UUUUUUUUUUUUUUUU	UUUUUUUUUUUUUUUU	EEEEEEEEEEEEEEEE	TTT	PPP	

UU	UU	EEEEEEEEEE	TTTTTTTTTT	DDDDDDDD	IIIIII	SSSSSSSS	KK	KK	000000	000000	
UU	UU	EEEEEEEEEE	TTTTTTTTTT	DDDDDDDD	IIIIII	SSSSSSSS	KK	KK	000000	000000	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UU	UU	EE	TT	DD	II	SS	KK	KK	00	00	
UUUUUUUUUU	UUUUUUUUUU	EEEEEEEEEE	TTTTTTTTTT	DDDDDDDD	IIIIII	SSSSSSSS	KK	KK	000000	000000	
UUUUUUUUUU	UUUUUUUUUU	EEEEEEEEEE	TTTTTTTTTT	DDDDDDDD	IIIIII	SSSSSSSS	KK	KK	000000	000000	

LL	IIIIII	SSSSSSSS	
LL	IIIIII	SSSSSSSS	
LL	II	SS	
LL	II	SS	
LL	II	SS	
LL	II	SS	
LL	II	SS	
LL	II	SS	
LL	II	SS	
LL	II	SS	
LL	II	SS	
LL	II	SS	
LL	II	SS	
LL	II	SS	
LL	II	SS	
LLLLLLLLLL	IIIIII	SSSSSSSS	
LLLLLLLLLL	IIIIII	SSSSSSSS	

(2)	101	Declarations
(4)	223	Read-Only Data
(5)	320	Read/Write Data
(6)	418	RMS-32 Data Structures
(7)	483	Test and Device Initialization
(10)	757	Test the Disk
(14)	912	Setup Routine for One File on a Unit
(16)	1006	Successful \$READ AST Routine
(19)	1163	Successful \$WRITE AST Routine
(20)	1210	Left Over File Cleanup Routine
(21)	1264	File Cleanup Routine
(22)	1311	Bum Data Routine
(23)	1370	Check if We've Stopped Testing a Unit
(24)	1421	\$CREATE-Specific Error Checker
(25)	1489	Timer Expiration Routine
(26)	1527	System Service Exception Handler
(27)	1657	General RMS Error Handler
(28)	1762	CTRL/C Handler
(29)	1809	Error Exit
(30)	1870	Exit Handler

```
0000 1 .TITLE UETDISK00 - VAX/VMS UETP DISK EXERCISER
0000 2 .IDENT 'V04-000'
0000 3 .ENABLE SUPPRESSION
0000 4 :
0000 5 :*****
0000 6 :*
0000 7 :* COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
0000 8 :* DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
0000 9 :* ALL RIGHTS RESERVED.
0000 10 :*
0000 11 :* THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
0000 12 :* ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
0000 13 :* INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
0000 14 :* COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
0000 15 :* OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
0000 16 :* TRANSFERRED.
0000 17 :*
0000 18 :* THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
0000 19 :* AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
0000 20 :* CORPORATION.
0000 21 :*
0000 22 :* DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
0000 23 :* SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
0000 24 :*
0000 25 :*
0000 26 :*****
0000 27 :
0000 28 :
0000 29 :++
0000 30 : FACILITY:
0000 31 : This module will be distributed with VAX/VMS under the [SYSTEST]
0000 32 : account.
0000 33 :
0000 34 : ABSTRACT:
0000 35 : Exercise testable units of a disk controller by writing and verifying
0000 36 : miscellaneous blocks of test files on those units.
0000 37 :
0000 38 : ENVIRONMENT:
0000 39 : This program will run in user access mode, with ASTs enabled except
0000 40 : during error processing. The program requires an AST limit of 55, a
0000 41 : buffered I/O limit of 55 and no privileges.
0000 42 :
0000 43 :--
0000 44 :
0000 45 : AUTHOR: Richard N. Holstein, CREATION DATE: August, 1981
0000 46 :
0000 47 : MODIFIED BY:
0000 48 :
0000 49 : V03-011 DAS0001 David Solomon 29-Feb-1984
0000 50 : Remove inappropriate reference to FAB EDL bit.
0000 51 :
0000 52 : V03-010 RNH0011 Richard N. Holstein, 15-Feb-1984
0000 53 : Take advantage of new UETP message codes. Fix SSERROR
0000 54 : interaction with RMS_ERROR. Fix ONE_SHOT_TEST register usage
0000 55 : for VERIFY_ERROR call.
0000 56 :
0000 57 : V03-009 RNH0010 Richard N. Holstein, 19-Dec-1983
```


0000 58 :
0000 59 :
0000 60 :
0000 61 :
0000 62 :
0000 63 :
0000 64 :
0000 65 :
0000 66 :
0000 67 :
0000 68 :
0000 69 :
0000 70 :
0000 71 :
0000 72 :
0000 73 :
0000 74 :
0000 75 :
0000 76 :
0000 77 :
0000 78 :
0000 79 :
0000 80 :
0000 81 :
0000 82 :
0000 83 :
0000 84 :
0000 85 :
0000 86 :
0000 87 :
0000 88 :
0000 89 :
0000 90 :
0000 91 :
0000 92 :
0000 93 :
0000 94 :
0000 95 :
0000 96 :
0000 97 :
0000 98 :
0000 99 : **

Give correct sentinels to Test Controller.

- V03-008 RNH0009 Richard N. Holstein, 29-Nov-1983
Be more specific when cleaning up test data files - works better
if multiple tests run simultaneously. Also when cleaning up,
terminate I/O cleanly to allow file erasure.
- V03-007 RNH0008 Richard N. Holstein, 11-Nov-1983
Use decimal conversion routine for unit numbers.
- V03-006 RNH0007 Richard N. Holstein, 05-Oct-1983
Get around an 11/750 Buffered Datapath bug which corrupts data,
and a driver patch which gives correct data at the expense of
data late errors being logged, by longword aligning the
read/write buffers. This is V03-001 in the V3 sources.
- V03-005 RNH0006 Richard N. Holstein, 26-May-1983
Exercise disks a bit more by setting the FOP EDL bit.
- V03-004 RNH0005 Richard N. Holstein, 10-Mar-1983
Invoke SSERROR in AST routines to prevent some improperly
handled conditions. Don't signal ending message in
EXIT_HANDLER.
- V03-003 RNH0004 Richard N. Holstein, 06-Jan-1983
Allow for longer file names and longer device names.
- V03-002 RNH0003 Richard N. Holstein, 15-Oct-1982
Miscellaneous fixes listed in the V3B UETP Workplan.
- V03-001 RNH0002 Richard N. Holstein, 06-Aug-1982
Edit to bug source only. See edit V03-006 above.
- V02-001 RNH0001 Richard N. Holstein, 11-Dec-1981
Allow further flexibility in testing by recovering from most
RMS\$_FUL situations when creating a file. Include PATTERN to
force a definite pattern to disk. Give a message when we've
completely stopped testing a unit. Suppress leading zeroes
when reading and remembering disk unit numbers. Clean up old
files before creating new ones.


```
0000 101 .SBTTL Declarations
0000 102 :
0000 103 : INCLUDE FILES:
0000 104 :
0000 105 : SYSSLIBRARY:LIB.MLB for general definitions
0000 106 : SHRLIB$:UETP.MLB for UETP definitions
0000 107 :
0000 108 : MACROS:
0000 109 :
0000 110 : $CHFDEF ; Condition handler frame definitions
0000 111 : $DEVDEF ; Device definitions
0000 112 : $DIBDEF ; Device Information Block
0000 113 : $SDVIDEF ; $GETDVI ITMLST item codes
0000 114 : $NAMDEF ; NAM block constants and offsets
0000 115 : $SHRDEF ; Shared messages
0000 116 : $SSDEF ; System Service status codes
0000 117 : $STSDEF ; Status return
0000 118 : $UETUNTDEF ; UETP unit block offset definitions
0000 119 : $UETPDEF ; UETP
0000 120 :
0000 121 : EQUATED SYMBOLS:
0000 122 :
0000 123 : Facility number definitions:
0000 124 : RMS$_FACILITY = 1
0000 125 :
0000 126 : SHR message definitions:
00740000 0000 127 : UETP = UETP$_FACILITY@STSSV FAC_NO ; Define the UETP facility code
007410E0 0000 128 : UETP$_ABENDD = UETP!SHR$_ABENDD ; Define the UETP message codes
00741038 0000 129 : UETP$_BEGIN = UETP!SHR$_BEGIN
00741080 0000 130 : UETP$_ENDEDD = UETP!SHR$_ENDEDD
00741098 0000 131 : UETP$_OPENIN = UETP!SHR$_OPENIN
00741130 0000 132 : UETP$_TEXT = UETP!SHR$_TEXT
0000 133 :
0000 134 : Internal flag bits...:
00000001 0000 135 : TEST_OVERV = 1 ; Set when test is over
00000002 0000 136 : SAFE_TO_UPDV = 2 ; Set if it's safe to update UETINIDEV
00000003 0000 137 : BEGIN_MSGV = 3 ; Set if 'BEGIN' msg has been printed
00000004 0000 138 : ONE_SHOTV = 4 ; Set if running in one-shot mode
00000005 0000 139 : PATTERNV = 5 ; Set if specific pattern data to write
0000 140 : ...and corresponding masks:
00000002 0000 141 : TEST_OVERM = 1@TEST_OVERV
00000004 0000 142 : SAFE_TO_UPDM = 1@SAFE_TO_UPDV
00000008 0000 143 : BEGIN_MSGM = 1@BEGIN_MSGV
00000010 0000 144 : ONE_SHOTM = 1@ONE_SHOTV
00000020 0000 145 : PATTERNM = 1@PATTERNV
0000 146 :
0000 147 : Miscellany:
00000020 0000 148 : LC_BITM = ^X20 ; Mask to convert lower case to upper
00000028 0000 149 : REC_SIZE = 40 ; UETINIDEV.DAT record size
00000084 0000 150 : TEXT_BUFFER = 132 ; Internal text buffer size
00000004 0000 151 : EFN2 = 4 ; EFN used for three minute timer
00000003 0000 152 : SS_SYNCH_EFN = 3 ; Synch miscellaneous system services
0000000F 0000 153 : MAX_PROC_NAME = 15 ; Longest possible process name
0000000A 0000 154 : MAX_DEV_DESIG = 10 ; Longest possible controller name
00000005 0000 155 : MAX_UNIT_DESIG = 5 ; Longest possible unit number
```



```
00000800 0000 157 ; Disk specific definitions:
00000800 0000 158 WRITE_SIZE = 512*4 ; RSZ buffer size
00000800 0000 159 READ_SIZE = WRITE_SIZE ; USZ buffer size (both use same buffer)
CCCD3E4C 0000 160 PERCENTAGE = ^F0.05 ; Percentage for incrementing files
00004040 0000 161 TOP_PCT = ^F0.75 ; Maximum percentage of unit we use
00000027 0000 162 FN_LEN = 39 ; Assumed maximum file name length
00000014 0000 163 BAKERS = 20 ; Baker's dozen: determine when to expand
55555555 0000 164 DISK_K_PATTERN = ^X55555555 ; Contents of test file FAB's CTX field
00000001 0000 165 DISK_K_CREATE = 1 ; Code to say current func is $CREATE
00000002 0000 166 DISK_V_NOCREATE = UETUNT$V_TESTABLE+1 ; On if $CREATE failed
00000004 0000 167 DISK_M_NOCREATE = 1@DISK_V_NOCREATE
00000003 0000 168 DISK_V_NOEXTEND = DISK_V_NOCREATE+1 ; On if $EXTEND failed
00000008 0000 169 DISK_M_NOEXTEND = 1@DISK_V_NOEXTEND
0000 170
0000 171 ;
0000 172 ; For each file on each unit, there will be a data structure set up, called a
0000 173 ; node. These nodes will be linked together in a self-relative queue whose
0000 174 ; header is UNIT_LIST. The first part of each node will be the standard
0000 175 ; definition from $UETUNTDEF. Following that will come the device test
0000 176 ; dependent stuff, defined below. NOTE THAT THIS DEFINITION IS DONE WITH AN
0000 177 ; ABSOLUTE PSECT. This means that what look like declarations are really
0000 178 ; definitions and the labels are really just offsets into a given node on the
0000 179 ; queue. (A not necessarily obvious consequence of using an ABS PSECT is that
0000 180 ; space must be reserved with .BLKx operations, since .BYTE, etc., attempt to
0000 181 ; store data.) This data structure differs from most other device tests in
0000 182 ; that there is more than one node per unit in normal and loop forever modes.
0000 183 ;
0000 184 .PSECT DEVDEP_STR_DEF,ABS,NOEXE,NOWRT,PAGE ; Note ABS attribute!
0000 185
000001A4 0000 186 .BLKB UETUNT$C_DEVDEP ; Skip over standard UETUNT block
000001A4 01A4 187
000001A6 01A4 188 DISK_W_UNIT: ; This unit number
000001A6 01A4 189 .BLKW 1
000001A6 01A6 190
000001A7 01A6 191 DISK_B_FILE: ; Number of this file on this unit
000001A7 01A6 192 .BLKB 1
000001A7 01A7 193
000001AB 01A7 194 DISK_L_OTHER_PTR: ; Ptr to other file node on this unit
000001AB 01A7 195 .BLKC 1
000001AB 01AB 196
000001AF 01AB 197 DISK_L_TOP_PCT: ; Maximum space we'll use on this unit
000001AF 01AB 198 .BLKL 1
000001AF 01AF 199
000001B3 01AF 200 DISK_L_SIZE: ; Cur. size of this file on this unit
000001B3 01AF 201 .BLKL 1
000001B3 01B3 202
000001BB 01B3 203 DISK_Q_FILEDESC: ; ASCID desc for the filespec...
000001BB 01B3 204 .BLKQ 1 ; ...actually used to create the file
000001BB 01BB 205
0000021B 01BB 206 DISK_C_NAM: ; NAM block for this file on this unit
0000021B 01BB 207 .BLKB NAM$C_BLN
0000021B 021B 208
0000031A 021B 209 DISK_T_NAMFILSPC: ; Resultant file spec per file per unit
0000031A 021B 210 .BLKB NAM$C_MAXRSS
0000031A 031A 211
00000176 031A 212 DEVDEP_SIZE = .-UETUNT$C_DEVDEP ; Device dependent part size of node
00000176 031A 213 ; Note that this excludes buffers
```


UETDISK00
V04-000

- VAX/VMS UETP DISK EXERCISER
Declarations

B 13

16-SEP-1984 01:22:47 VAX/VMS Macro V04-00
5-SEP-1984 04:24:57 [UETP.SRC]UETDISK00.MAR;1

Page 5
(3)

```

      031A 214
      031A 215
      031C 216 DISK_K_BUFFER: .ALIGN LONG ; Prevent DDP data lates on 11/750 UDA
00000B1C 031C 217 .BLKB WRITE_SIZE ; I/O buffer for both reads and writes
      0B1C 218
      0B1C 219
      0B1C 220
00000006 0B1C 221
      PAGES = <<UETUNT$C_INDSIZ+- ; Add together all of the pieces...
              DEVDEP_SIZE+- ; ...which make up a UETP unit block...
              WRITE_SIZE+511>/512> ; ...to give to the $EXPREG service
```



```
00000000 0B1C 223 .SBTTL Read-Only Data
00000000 224 .PSECT RODATA,NOEXE,NOWRT,PAGE
0000 225
0000 226 ACNT_NAME: ; Process name on exit
53 45 54 53 59 53 00000008'010E0000' 0000 227 .ASCID /SYSTEST/
54 000E
000F 228
000F 229 TEST_NAME: ; This test name
53 49 44 54 45 55 00000017'010E0000' 000F 230 .ASCID /UETDISK00/
30 30 4B 001D
0020 231
0020 232 SUPDEV_GBLSEC: ; How we access UETSUPDEV.DAT
50 55 53 54 45 55 00000028'010E0000' 0020 233 .ASCID /UETSUPDEV/
56 45 44 002E
0031 234
0031 235 CONTROLLER: ; Logical name of controller
41 4E 4C 52 54 43 00000039'010E0000' 0031 236 .ASCID /CTRLNAME/
45 4D 003F
0041 237
0041 238 MODE: ; Run mode logical name
45 44 4F 4D 00000049'010E0000' 0041 239 .ASCID /MODE/
004D 240
004D 241 NO_RMS_AST_TABLE: ; List of errors for which...
00000000' 004D 242 .LONG RMSS_BLN ; ...RMS cannot deliver an AST...
00000000' 0051 243 .LONG RMSS_BUSY ; ...even if one has an ERR= arg
00000000' 0055 244 .LONG RMSS_CDA ; Note that we can search table...
00000000' 0059 245 .LONG RMSS_FAB ; ...via MATCHC since <31:16>...
00000000' 005D 246 .LONG RMSS_RAB ; ...pattern can't be in <15:0>
00000014 0061 247 NRAT_LENGTH = .-NO_RMS_AST_TABLE
0061 248
0061 249 SYSS$INPUT: ; Name of device from which...
4E 49 24 53 59 53 00000069'010E0000' 0061 250 .ASCID /SYSS$INPUT/
54 55 50 006F ; ...the test can be aborted
0072 251
0072 252 INPUT_ITMLST: ; $GETDVI arg list for SYSS$INPUT
0020 0040 0072 253 .WORD 64,DVIS$ DEVNAM ; We need the equivalence name
0000000C'00000014' 0076 254 .LONG BUFFER,BUFFER_PTR
00000000 007E 255 .LONG 0 ; Terminate the list
0082 256
0082 257 CS1: ; Device class and type control string
21 20 42 58 32 21 0000008A'010E0000' 0082 258 .ASCID /!2XB !2XB /
20 42 58 32 0090
0094 259
0094 260 CS3: ; Device class-only control string
2A 20 42 58 32 21 0000009C'010E0000' 0094 261 .ASCID /!2XB **/
2A 00A2
00A3 262
00A3 263 CNTRLMSG:
65 74 72 6F 62 41 000000AB'010E0000' 00A3 264 .ASCID \Aborted via a user CTRL/C\
72 65 73 75 20 61 20 61 69 76 20 64 00B1
43 2F 4C 52 54 43 20 00BD
00C4 265
00C4 266 NO_CTRLNAME:
6E 6F 63 20 6F 4E 000000CC'010E0000' 00C4 267 .ASCID /No controller specified./
63 65 70 73 20 72 65 6C 6C 6F 72 74 00D2
2E 64 65 69 66 69 00DE
00E4 268
```



```
20 74 27 6E 61 43 000000EC'010E0000' 00E4 269 DEAD_CTRLNAME:
6C 6F 72 74 6E 6F 63 20 74 73 65 74 00E4 270 .ASCID /Can't test controller !AS, marked as unusable in UETINIDEV.DAT./
72 61 6D 20 2C 53 41 21 20 72 65 6C 00F2
61 73 75 6E 75 20 73 61 20 64 65 6B 00FE
4E 49 54 45 55 20 6E 69 20 65 6C 62 010A
2E 54 41 44 2E 56 45 44 49 0116
0122
012B 271
012B 272 NOUNIT_SELECTED:
012B 273 .ASCID /No units selected for testing./
0139
0145
0151 274
0151 275 ILLEGAL_REC:
0151 276 .ASCID /Illegal record format in file UETINIDEV.DAT!/
015F
016B
0177
0183
0185 277
0185 278 PASS_MSG:
0193 279 .ASCID /End of pass !UL with !UL iterations at !%D./
019F
01AB
01B7
01B8 280
01B8 281 INIDEV_UPDERR: ; Error during exit handler
01C6 282 .ASCID /Error updating UETINIDEV.DAT./
01D2
01DD 283
01DD 284 THREEMIN: ; 3 minute delta time
01DD 285 .LONG -10*1000*1000*180,-1
01E5 286
01E5 287 UNIT_DESC: ; Descriptor used to convert unit #
01E9 288 .LONG 5
01E9 289 .ADDRESS BUFFER+6
01ED 290
01ED 291 CONT_DESC: ; Descriptor used to convert controller...
01ED 292 .WORD REC_SIZE,0 ; ...from lowercase to uppercase
01F1 293 .ADDRESS BUFFER
01F5 294
01F5 295 FILE: ; Fills in RMS_ERR_STRING
0201 296 .ASCID /file/
0201 297
0201 298 RECORD: ; Fills in RMS_ERR_STRING
020F 299 .ASCID /record/
020F 300
020F 301 RMS_ERR_STRING: ; Announces an RMS error
021D 302 .ASCID /RMS !AS error in file !AD/
0229
0230 303
0230 304 PROMPT:
0230 305 .ASCII /Controller designation?: /
023C
```

66 6F 20 64 6E 45 0000018D'010E0000' 0185
69 77 20 4C 55 21 20 73 73 61 70 20 0193
61 72 65 74 69 20 4C 55 21 20 68 74 019F
44 25 21 20 74 61 20 73 6E 6F 69 74 01AB
2E 01B7
01B8
20 72 6F 72 72 45 000001C0'010E0000' 01B8
54 45 55 20 67 6E 69 74 61 64 70 75 01C6
2E 54 41 44 2E 56 45 44 49 4E 49 01D2
01DD
FFFFFFFF 94B62E00 01DD
01DD 284 THREEMIN: ; 3 minute delta time
01DD 285 .LONG -10*1000*1000*180,-1
01E5 286
01E5 287 UNIT_DESC: ; Descriptor used to convert unit #
01E9 288 .LONG 5
01E9 289 .ADDRESS BUFFER+6
01ED 290
01ED 291 CONT_DESC: ; Descriptor used to convert controller...
01ED 292 .WORD REC_SIZE,0 ; ...from lowercase to uppercase
01F1 293 .ADDRESS BUFFER
01F5 294
01F5 295 FILE: ; Fills in RMS_ERR_STRING
0201 296 .ASCID /file/
0201 297
0201 298 RECORD: ; Fills in RMS_ERR_STRING
020F 299 .ASCID /record/
020F 300
020F 301 RMS_ERR_STRING: ; Announces an RMS error
021D 302 .ASCID /RMS !AS error in file !AD/
0229
0230 303
0230 304 PROMPT:
0230 305 .ASCII /Controller designation?: /
023C

65 6C 69 66 000001FD'010E0000' 01F5
64 72 6F 63 65 72 00000209'010E0000' 0201
41 21 20 53 4D 52 00000217'010E0000' 020F
66 20 6E 69 20 72 6F 72 72 65 20 53 021D
44 41 21 20 65 6C 69 0229


```

      20 0248
00000019 0249 306          PMTSIZ = .-PROMPT
      0249 307
      0249 308 SYSTEST_DIR:          ; Part of filespec for test files...
5D 54 53 45 54 53 59 53 5B 00' 0249 309          .ASCIC  /[SYSTEST]/      ; ...if they're on a scratch pack
      09 0249
      0253 310
      0253 311 SYS0_SYSTEST_DIR:      ; Part of filespec for test files...
45 54 53 59 53 2E 30 53 59 53 5B 00' 0253 312          .ASCIC  /[SYS0.SYSTEST]/  ; ...if they're on a V3 system pack
      5D 54 53 025F
      0E 0253
      0262 313
      0262 314 DOTTST:                ; Part of filespec for test files
      54 53 54 2E 00' 0262 315          .ASCIC  /.TST/
      04 0262
      0267 316
      0267 317 PATTERN:                ; Logical name to specify pattern...
52 45 54 54 41 50 0000026F'010E0000' 0267 318          .ASCID  /PATTERN/      ; ...to write to test files
      4E 0275
```



```
0276 320 .SBTTL Read/Write Data
00000000 321 .PSECT RWDATA,WRT,NOEXE,PAGE
0000 322
0000 323 TTCHAN: ; Channel associated with ctrl. term.
0000 324 .WORD 0
0002 325
0000 326 FLAG: ; Miscellaneous flag bits
0000 327 .WORD 0 ; (See Equated Symbols for definitions)
0004 328
0000 329 FAO_BUF: ; FAO output string descriptor
0000 0084 0004 330 .WORD TEXT_BUFFER,0
00000014' 0008 331 .ADDRESS BUFFER
000C 332
0000 0084 000C 333 BUFFER_PTR: ; Fake .ASCID buffer for misc. strings
00000014' 0010 334 .WORD TEXT_BUFFER,0 ; A word for length, a word for desc.
0014 335 .ADDRESS BUFFER
0014 336
00000098 0014 337 BUFFER: ; FAO output and other misc. buffer
0098 338 .BLKB TEXT_BUFFER
0098 339
0000 000A 0098 340 DEVDESC: ; Device name descriptor
000000B7' 009C 341 .WORD MAX_DEV_DESIG,0
00A0 342 .ADDRESS DEV_NAME
00A0 343
00000008' 00A0 344 PROCESS_NAME: ; Process name
00000008' 00AC 345 .ASCID /DISK/
000000B7' 00AC 346 PROCESS_NAME_FREE = MAX_PROC_NAME-<.-8-PROCESS_NAME>
00B7 347 .BLKB PROCESS_NAME_FREE
00B7 348
000000C6 00B7 349 DEV_NAME: ; Device name buffer
0000000F' 00C6 350 .BLKB MAX_DEV_DESIG+MAX_UNIT_DESIG
00C6 351 NAME_LEN = .-DEV_NAME
00C6 352
0000 0074 00C6 353 DIB: ; Device Information Block
000000CE' 00CA 354 .WORD DIB$K_LENGTH,0
00000142 00CE 355 .ADDRESS DIBBUF
00142 356 DIBBUF:
00142 357 .BLKB DIB$K_LENGTH
00000000 0142 358
00142 359 ERROR_COUNT: ; Cumulative error count at runtime
0146 360 .LONG 0
0146 361
00000000 0146 362 STATUS: ; Status value on program exit
014A 363 .LONG 0
014A 364
00000000 00000000 014A 365 QUAD_STATUS: ; IO status block for misc sys. svcs.
0152 366 .QUAD 0
0152 367
00000000 00000000 0152 368 INADDRESS: ; $CRMPSC address storage
015A 369 .LONG 0,0
00000000 00000000 015A 370 OUTADDRESS:
0162 371 .LONG 0,0
0162 372
0000 0162 373 UNIT_NUMBER: ; Current dev unit number
0164 374 .WORD 0
0164 375
0164 376 DEVNAM_LEN: ; Current device name length
```



```
0000 0164 377 .WORD 0
      0166 378
      0166 379 ; RANDOM1 and RANDOM2 may be combined to produce a set of pseudo-random numbers
      0166 380 RANDOM1: ; Random word #1
AAAAAAA 0166 381 .LONG ^XAAAAAAAA
      016A 382
      016A 383 RANDOM2: ; Random word #2
A72EA72E 016A 384 .LONG ^XA72EA72E
      016E 385
      016E 386 ITERATION: ; # of times all tests were executed
00000000 016E 387 .LONG 0
      0172 388
      0172 389 PASS: ; Pass count
00000000 0172 390 .LONG 0
      0176 391
      0176 392 MSG_BLOCK: ; Auxiliary $GETMSG info
0000017A 0176 393 .BLKB 4
      017A 394
      017A 395 EXIT_DESC: ; Exit handler descriptor
00000000 017A 396 .LONG 0
00000C1C 017E 397 .ADDRESS EXIT_HANDLER
00000001 0182 398 .LONG 1
00000146 0186 399 .ADDRESS STATUS
      018A 400
      018A 401 ARG_COUNT: ; Argument counter used by ERROR_EXIT
00000000 018A 402 .LONG 0
      018E 403
      018E 404 .ALIGN QUAD ; For self-relative queue of unit blocks
      0190 405
      0190 406 UNIT_LIST: ; Head of unit block circular list
00000000 00000000 0190 407 .QUAD 0
      0198 408
      0198 409 NEW_NODE: ; Newly acquired node address
00000000 00000000 0198 410 .QUAD 0
      01A0 411
      01A0 412 RANDOM_DATA: ; Buffer to fill with "random" data
000009A0 01A0 413 .BLKL <WRITE_SIZE+3>/4 ; Ensure it's a multiple of longwords
      09A0 414
      09A0 415 PATTERN_DATA: ; Holds pattern to write to test...
000009A4 09A0 416 .BLKL 1 ; ...files if PATTERN is defined
```



```
09A4 418 .SBTTL RMS-32 Data Structures
09A4 419 .ALIGN LONG
09A4 420
09A4 421 SYSIN_FAB: ; Allocate FAB for SYSS$INPUT
09A4 422 $FAB-
09A4 423 FNM = <SYSS$INPUT>
09F4 424
09F4 425 SYSIN_RAB: ; Allocate RAB for SYSS$INPUT
09F4 426 $RAB-
09F4 427 FAB = SYSIN_FAB,-
09F4 428 ROP = PMT,-
09F4 429 PBF = PROMPT,-
09F4 430 PSZ = PMTSIZ,-
09F4 431 UBF = DEV_NAME,-
09F4 432 USZ = NAME_LEN
0A38 433
0A38 434 INI_FAB: ; Allocate FAB for UETINIDEV
0A38 435 $FAB-
0A38 436 FAC = <GET,PUT,UPD>,-
0A38 437 RAT = CR,-
0A38 438 SHR = <GET,PUT,UPI>,-
0A38 439 FNM = <UETINIDEV.DAT>
0A88 440
0A88 441 INI_RAB: ; Allocate RAB for UETINIDEV
0A88 442 $RAB-
0A88 443 FAB = INI_FAB,-
0A88 444 RBF = BUFFER,-
0A88 445 UBF = BUFFER,-
0A88 446 USZ = REC_SIZE
0ACC 447
0ACC 448 DDB_RFA: ; RFA storage for INI_RAB
00000AD2 0ACC 449 .BLKB 6
0AD2 450
0AD2 451 .ALIGN LONG
0AD4 452 SUP_FAB: ; Allocate FAB for UETSUPDEV
0AD4 453 $FAB-
0AD4 454 FAC = GET,-
0AD4 455 SHR = <UPI,GET>,-
0AD4 456 RAT = CR,-
0AD4 457 FOP = UFO,-
0AD4 458 FNM = <UETSUPDEV.DAT>
0B24 459 ;
0B24 460 ; Dummy FAB and RAB to copy to the UETP unit blocks
0B24 461 ; The following FAB and RAB must be contiguous and in this order!
0B24 462 ;
0B24 463
0B24 464 DUMMY_FAB:
0B24 465 $FAB-
0B24 466 CTX = DISK_K PATTERN,- ; This will identify a test file
0B24 467 FAC = <BIO,POT,GET>,-
0B24 468 ; FOP = <EDL>,-
0B24 469 MRS = WRITE_SIZE,-
0B24 470 ORG = SEQ,-
0B24 471 RFM = UDF
0B74 472
0B74 473 DUMMY_RAB:
0B74 474 $RAB-
```

0B74	475	ROP = <ASY,BIO>,-
0B74	476	RSZ = WRITE_SIZE,-
0B74	477	USZ = READ_SIZE
0BB8	478	
0BB8	479	DUMMY_NAM:
0BB8	480	\$NAM-
0BB8	481	RSS = NAMSC_MAXRSS

; RAB\$M_ASY cleared in one-shot mode

; More specifically identify test file


```
00000000 483 .SBTTL Test and Device Initialization
00000000 484 .PSECT DISK,EXE,NOWRT,PAGE
00000000 485
00000000 486 .DEFAULT DISPLACEMENT,WORD
00000000 487
00000000 488 ;+
00000000 489 ;
00000000 490 ; Start up the disk test. This entails some overhead necessary to cope
00000000 491 ; with both expected and unforeseen conditions, figuring out just what
00000000 492 ; devices are to be tested, making sure we can test the indicated devices
00000000 493 ; and setting up writeable space for each device to be tested.
00000000 494 ;-
00000000 495 .ENTRY UETDISK00,^M<> ; Entry mask
00000000 496
6D 09A5'CF DE 0002 497 MOVAL SSERROR,(FP) ; Declare exception handler
00000000 498 $SETSFMS ENBFLG = #1 ; Enable system service failure mode
00000000 499 $DCLEXH_S DESBLK = EXIT_DESC ; Declare an exit handler
00000000 500
00000000 501 $OPEN FAB = SYSIN FAB,- ; Open SYS$INPUT
00000000 502 ERR = RMS ERROR
00000000 503 $CONNECT RAB = SYSIN RAB,- ; Connect RAB to SYS$INPUT
00000000 504 ERR = RMS ERROR
00000000 505 BBC S^#DEV$V TRM,- ; BR if SYS$INPUT is NOT a terminal
00000000 506 SYSIN FAB+FAB$ DEV,10$
00000000 507 $TRNLOG_S LOGNAM = CONTROLLER,- ; Allow terminal user to specify...
00000000 508 RSLLEN = DEVNAM_LEN,- ; ...a logical name...
00000000 509 RSLBUF = DEVNSC ; ...for the controller to test
00000000 510 CMPL RO,#SS$ NORMAL ; Was a controller specified?
00000000 511 BEQL PROC_CONT_NAME ; BR if it was - go process it
00000000 512 10$:
00000000 513 $GET RAB = SYSIN RAB,- ; Read SYS$INPUT...
00000000 514 ERR = RMS ERROR ; ...for controller name
00000000 515 MOVW SYSIN RAB+RAB$W_RSZ,- ; Save the name length
00000000 516 DEVNAM_LEN
00000000 517 BNEQ PROC_CONT_NAME ; BR if we got something
00000000 518 MOVL #SS$BADPARAM,STATUS ; Save an exit status if not
00000000 519 PUSHAL NO_CTRLNAME ; Prepare for message...
00000000 520 PUSHL #1 ; ...
00000000 521 PUSHL #UETP$_TEXT!STSSK_ERROR ; ...
00000000 522 PUSHL #3 ; ...
00000000 523 BRW ERROR_EXIT ; ...to tell of bad setup
00000000 524
00000000 525 PROC_CONT_NAME:
00000000 526 MOVZWL DEVNAM_LEN,DEVNSC ; Set the device name length
00000000 527 PUSHAL DEVNSC ; Make sure...
00000000 528 PUSHAL DEVNSC ; ...that the specified controller...
00000000 529 CALLS #2,G^STR$UPCASE ; ...is all uppercase for later comparison
00000000 530 ADDL3 #1,DEVNSC,R2 ; Estimate the eventual...
00000000 531 ADDW2 R2,PROCESS_NAME ; ...process name length (incl. '"')
00000000 532 MOVAL PROCESS_NAME+8- ; Locate first available byte...
00000000 533 +MAX_PROC_NAME- ; ...in process name handle...
00000000 534 -PROCESS_NAME_FREE,RO ; ...for device name
00000000 535 SUBL3 #PROCESS_NAME_FREE,- ; Will the device name fit...
00000000 536 R2,R1 ; ...in the remaining space?
00000000 537 BLEQ 10$ ; BR if it will
00000000 538 SUBL2 R1,RO ; Overwrite handle otherwise...
00000000 539 MOVW #MAX_PROC_NAME,PROCESS_NAME ; ...and define the maximum length
```



```
60      80      5F 8F      90      00BF      540 10$:
00B7'CF      0098'CF      28      00C3      541
              7E      D4      00CB      542
              000F'CF      DF      00CD      543
              02      DD      00D1      544
              00741039 8F      DD      00D3      545
00000000'GF      04      FB      00D9      546
              0002'CF      08      A8      00E0      547
              00E5      548
              00F0      549
              02      E1      00F0      550
66 09E4'CF      00F2      551
              00F6      552
              00F6      553
              00F6      554
              00F6      555
              00F6      556
45 014A'CF      E9      0112      557
              0117      558
              0117      559
              0128      560
              0128      561
              0128      562
              00A0'CF      DF      0149      563
              01      DD      014D      564
              0074832B 8F      DD      014F      565
00000000'GF      03      FB      0155      566
              015C      567 20$:
              015C      568
              015C      569
              015C      570
              0014'CF      20      8A      0175      571
0014'CF      4F 8F      91      017A      572
              0A      12      0180      573
              0002'CF      10      A8      0182      574
              0B78'CF      01      CA      0187      575
              018C      576 25$:
              018C      577
              018C      578
              018C      579
              17 50      E9      01A5      580
              09A0'CF      DF      01A8      581
              000C'CF      DF      01AC      582
00000000'GF      02      FB      01B0      583
              05 50      E9      01B7      584
              0002'CF      20      A8      01BA      585
              01BF      586 27$:

MOV B      #A/ /,(R0)+      ; Separate handle from device name
MOV C3     DEV DSC,DEV_NAME,(R0)      ; Concatenate handle with device name
CLRL      -(SP)      ; Set the time stamp flag
PUSHAL     TEST_NAME      ; Set the test name
PUSHL      #2      ; Push the argument count
PUSHL      #UETP$_BEGIN!STSSK_SUCCESS      ; Set the message code
CALLS      #4,G^LIB$SIGNAL      ; Print the startup message
BISW2      #BEGIN MSGM,FLAG      ; Set flag so we don't print it again
$SETPRN_S  PRCNAM = PROCESS_NAME      ; Set the process name to UETDISK00_x

BBC      S^#DEV$V TRM,-      ; BR if SYSS$INPUT is NOT a terminal
          SYSIN FAB+FAB$DEV,20$
$GETDVI_S  DEVNAM = SYSS$INPUT,-      ; Get the name of...
          EFN = #SS SYNCH EFN,-      ; ...device which may abort test
          ITMLST = INPUT ITMLST,-
          IOSB = QUAD STATUS
BLBC      QUAD STATUS,20$      ; Avoid CTRL/C handler if any error
$ASSIGN_S  DEVNAM = BUFFER_PTR,-      ; Set up for CTRL/C AST handling
          CHAN = TTCHAN
$QIOW_S    CHAN = TTCHAN,-      ; Enable CTRL/C AST's...
          FUNC = #IOS SETMODE!IOSM_CTRLCAST,-
          P1 = CCASTHAND
PUSHAL     PROCESS_NAME      ; ...and tell the user...
PUSHL      #1
PUSHL      #UETP$_ABORTC!STSSK_SUCCESS      ; ...how to abort gracefully...
CALLS      #3,G^LIB$SIGNAL      ; ...

$STRNLOG_S LOGNAM = MODE,-      ; Get the run mode
          RSLLEN = BUFFER_PTR,-
          RSLBUF = FAO_BUF
BICB2      #LC BITM,BUFFER      ; Convert to upper case
CMPB      #A70/,BUFFER      ; Is this a one shot?
BNEQ      25$      ; BR if not
BISW2      #ONE_SHOTM,FLAG      ; Set flag for one-shot mode...
BICL2      #RAB$M_ASY,DUMMY_RAB+RAB$L_ROP      ; ...and clear asynchronous flag

$STRNLOG_S LOGNAM = PATTERN,-      ; Do we have a specific pattern...
          RSLLEN = BUFFER_PTR,-      ; ...to write to all buffers?
          RSLBUF = FAO_BUF
BLBC      R0,27$      ; BR if we don't
PUSHAL     PATTERN_DATA      ; We may...
PUSHAL     BUFFER_PTR
CALLS      #2,G^OTSS$CVT_TZ_L      ; ...convert ASCII pattern to binary
BLBC      R0,27$      ; BR if there's an error
BISW2      #PATTERNM,FLAG      ; Indicate the presence of a pattern
```



```
01BF 588 :  
01BF 589 : From UETINIDEV.DAT and UETSUPDEV.DAT, get information which gives controller  
01BF 590 : and unit configuration and lets us know if the setup to run this test was  
01BF 591 : done correctly. Allocate space for each file on a unit.  
01BF 592 :  
01BF 593 $OPEN FAB = INI_FAB,- ; Open file 'UETINIDEV.DAT'  
01BF 594 ERR = RMS_ERROR  
01CE 595 $CONNECT RAB = INI_RAB,- ; Connect the RAB and FAB  
01CE 596 ERR = RMS_ERROR  
01DD 597 $MGBLSC_S INADR = INADDRESS,- ; Connect to UETSUPDEV global section  
01DD 598 RETADR = OUTADDRESS,-  
01DD 599 GSDNAM = SUPDEV_GBLSEC,-  
01DD 600 FLAGS = #SEC$M_EXPREG  
00000978 8F 50 D1 01FC 601 CMPL RO,#SS$_NOSUCHSEC ; Was the section already there?  
37 12 0203 602 BNEQ 30$ ; BR if it was...  
0205 603 $OPEN FAB = SUP_FAB,- ; ...else open 'UETSUPDEV.DAT'  
0205 604 ERR = RMS_ERROR  
0214 605 $CRMPSC_S CHAN = SUP_FAB+FAB$_STV,- ; Create the global section  
0214 606 INADR = INADDRESS,-  
0214 607 RETADR = OUTADDRESS,-  
0214 608 GSDNAM = SUPDEV_GBLSEC,-  
0214 609 FLAGS = #SEC$M_EXPREG!SEC$M_GBL  
56 015E'CF 015A'CF C3 023C 610 30$: SUBL3 OUTADDRESS,OUTADDRESS+4,R6 ; Compute global section length  
023C 611  
0244 612  
0244 613 FIND_IT:  
0244 614 $GET RAB = INI_RAB,- ; Get the first record  
0244 615 ERR = RMS_ERROR  
01ED'CF DF 0253 616 PUSHAL CONT_DESC ; Make sure...  
01ED'CF DF 0257 617 PUSHAL CONT_DESC ; ...that the controller name...  
00000000'GF 02 FB 025B 618 CALLS #2,G*STR$UPCASE ; ...is all uppercase letters  
0014'CF 44 8F 91 0262 619 CMPB #^A/D/,BUFFER ; Is this a DDB?  
27 13 0268 620 BEQL 10$ ; Go on if not  
0014'CF 45 8F 91 026A 621 CMPB #^A/E/,BUFFER ; Is this the end of the file?  
D2 12 0270 622 BNEQ FIND_IT ; Continue on if not  
0098'CF DF 0272 623 PUSHAL DEV$DC ; Push device not supported message  
00A0'CF DF 0276 624 PUSHAL PROCESS_NAME ; Parameters on the stack  
02 DD 027A 625 PUSHL #2  
00748333 8F DD 027C 626 PUSHL #UETPS_DENOSU  
02 F0 0282 627 INSV #ST$K_ERROR,- ; Set the severity code...  
00 0284 628 #ST$V_SEVERITY,-  
6E 03 0285 629 #ST$S_SEVERITY,(SP)  
0146'CF 6E D0 0287 630 MOVL (SP),STATUS ; ...and save it as the exit status  
04 DD 028C 631 PUSHL #4  
090B 31 028E 632 BRW ERROR_EXIT ; Exit in error  
0291 633 10$:  
00B7'CF 001A'CF 0164'CF 29 0291 634 CMPC DEVDNAM_LEN,BUFFER+6,DEV_NAME ; Is this the right controller?  
A7 12 029B 635 BNEQ FIND_IT ; BR if not  
OACC'CF OA98'CF 06 28 029D 636 MOVC3 #6,INI_RAB+RAB$_RFA,DDB_RFA ; Save the record file address  
0018'CF 54 8F 91 02A5 637 CMPB #^A/T/,BUFFER+4 ; Can we test this controller?  
2F 13 02AB 638 BEQL FOUND_IT ; BR if we can...  
02AD 639 $FAO_S CTRSTR = DEAD_CTRLNAME,- ; ...and yell at user if we can't  
02AD 640 OUTLEN = BUFFER_PTR,-  
02AD 641 OUTBUF = FAO_BUF,-  
02AD 642 P1 = #DEV$DC  
0146'CF 14 D0 02C6 643 MOVL #SS$_BADPARAM,STATUS ; Set return status  
000C'CF DF 02CB 644 PUSHAL BUFFER_PTR ; ...
```



```
00741132 01 DD 02CF 645      PUSHL #1
00741132 8F DD 02D1 646      PUSHL #UETP$_TEXT!STSSK_ERROR ; ...
00741132 03 DD 02D7 647      PUSHL #3 ; ...
08C0 31 02D9 648      BRW ERROR_EXIT ; We can't test what we can't test
02DC 649
02DC 650 FOUND_IT:
02DC 651 $GET RAB = INI_RAB,- ; Get a record
02DC 652 ERR = RMS_ERROR ; ...that this line...
01ED'CF DF 02EB 653 PUSHAL CONT_DESC ; Make sure...
01ED'CF DF 02EF 654 PUSHAL CONT_DESC ; ...is all uppercase letters
00000000'GF 02 FB 02F3 655 CALLS #2,G*STR$UPCASE ; ...is all uppercase letters
0014'CF 55 8F 91 02FA 656 CMPB #^A/U/,BUFFER ; Is this a UCB?
0014'CF 24 13 0300 657 BEQL 30$ ; BR if it is
0014'CF 44 8F 91 0302 658 CMPB #^A/D/,BUFFER ; Is this a DDB?
0014'CF 19 13 0308 659 BEQL 20$ ; BR if yes
0014'CF 45 8F 91 030A 660 CMPB #^A/E/,BUFFER ; Is this the end?
0014'CF 11 13 0310 661 BEQL 20$ ; BR if yes
0312 662 10$:
0151'CF DF 0312 663 PUSHAL ILLEGAL_REC ; Then this is an error in the record
0151'CF 01 DD 0316 664 PUSHL #1 ; Push the error message
00741132 8F DD 0318 665 PUSHL #UETP$_TEXT!STSSK_ERROR ; Push the signal name
00741132 03 DD 031E 666 PUSHL #3 ; Push the temp arg count
0879 31 0320 667 BRW ERROR_EXIT ; Finish for good
0323 668 20$:
00EE 31 0323 669 BRW ALL_SET ; Found DDB or END
0326 670 30$:
0018'CF 54 8F 91 0326 671 CMPB #^A/T/,BUFFER+4 ; Is the unit testable?
0018'CF AE 12 032C 672 BNEQ FOUND_IT ; BR if not
0018'CF 01 DD 032E 673 PUSHL #1 ; Flag to ignore blanks when converting
0018'CF 02 DD 0330 674 PUSHL #2 ; Set byte size of results
0162'CF DF 0332 675 PUSHAL UNIT_NUMBER ; Set address to receive word
01E5'CF DF 0336 676 PUSHAL UNIT_DESC ; Push string address
00000000'GF 04 FB 033A 677 CALLS #4,G*OT$SCVT_TI_L ; Convert ASCII unit # to decimal
00000000'GF CE 50 E9 0341 678 BLBC R0,10$ ; Don't allow a bogus unit to pass
00000000'GF 05 20 3B 0344 679 SKPC #^A/ /,MAX_UNIT_DESIG,- ; Find out where unit number really is
001A'CF 50 D7 034A 680 DECL R0 ; Units must all be at least one digit
001A'CF 61 50 3B 034C 681 SKPC #^A/O/,R0,(R1) ; Skip leading zeroes on the unit
001A'CF 50 D6 0350 682 INCL R0 ; Compensate for DECL above
0098'CF 0164'CF 50 A1 0352 683 ADDW3 R0,DEVNAM_LEN,DEVDSK ; Calculate device unit string length
0098'CF 52 0164'CF 3C 035A 684 MOVZWL DEVNAM_LEN,R2 ; Offset to unit number in DEVDSK
00B7'C2 61 50 28 035F 685 MOVCS R0,(R1),DEV_NAME(R2) ; Append unit number to device
00B7'C2 0365 686 $GETDEV_S DEVNAM = DEVDSK,- ; Get the device characteristics
00B7'C2 0365 687 PRIBUF = DIB
00B7'C2 57 00D2'CF 9A 037A 688 MOVZBL DIBBUF+DIB$B_DEVCLASS,R7 ; Save the device class
00B7'C2 58 00D3'CF 9A 037F 689 MOVZBL DIBBUF+DIB$B_DEVTYPE,R8 ; Save the device type
00B7'C2 0384 690 $FAO_S CTRSTR = CS1,-
00B7'C2 0384 691 OUTBUF = FAO_BUF,-
00B7'C2 0384 692 P1 = R7,-
00B7'C2 0384 693 P2 = R8 ; Make it into a string
015A'DF 56 0014'CF 06 39 0399 694 MATCHC #6,BUFFER,R6,@OUTADDRESS ; Find the device class and type
015A'DF 1E 13 03A2 695 BEQL 40$ ; BR if it was found
015A'DF 03A4 696 $FAO_S CTRSTR = CS3,- ; Try for full class support
015A'DF 03A4 697 OUTBUF = FAO_BUF,-
015A'DF 03A4 698 P1 = R7
015A'DF 56 0014'CF 06 39 03B7 700 MATCHC #6,BUFFER,R6,@OUTADDRESS ; Find the device class only
015A'DF 0D 12 03C0 701 BNEQ 50$ ; BR if not found
```



```

55 000F'CF 9A 03C2 702 40$:
0017'CF 63 55 29 03C2 703
1F 13 03C7 704
0098'CF DF 03CD 705
00A0'CF DF 03CF 706 50$:
02 DD 03CF 707
00748333 8F DD 03D3 708
02 DD 03D7 709
00 03D9 710
03 03DF 711
0146'CF 03 03E1 712
6E 03 03E2 713
04 DD 03E4 714
07AE 31 03E9 715
03EB 716 60$:
00 DD 03EE 717
060B'CF 01 FB 03F0 718
16 0002'CF 04 E0 03F5 719
50 DD 03FB 720
01 DD 03FD 721
060B'CF 01 FB 03FF 722
51 8ED0 0404 723
01A7 C1 50 D0 0407 724
01A7 C0 51 D0 040C 725
FEC8 31 0411 726 70$:
0411 727
0411 728
MOVZBL TEST_NAME,R5 ; Get the test name length
CMPC3 R5,(R3),TEST_NAME+8 ; Are we the right test?
BEQL 60$ ; BR if yes
PUSHAL DEVDSO ; Push device not supported message
PUSHAL PROCESS_NAME ; Parameters on the stack
PUSHL #2 ; Push the argument count
PUSHL #UETP$_DENOSU
INSV #STSSK_ERROR,-
#STSSV_SEVERITY,-
#STSS$SEVERITY,(SP) ; Set the severity code...
MOVL (SP),STATUS ; ...and save it as the exit status
PUSHL #4 ; Push the partial arg count...
BRW ERROR_EXIT ; ...and split this scene
PUSHL #0 ; Allocate and initialize...
CALLS #1,UNIT_FILE_SETUP ; ...space for file 0 on this unit
BBS #ONE_SHOTV,FLAG,70$ ; Omit file 1 if one-shot mode
PUSHL R0 ; Save pointer so file 1 can point here
PUSHL #1 ; Allocate and initialize...
CALLS #1,UNIT_FILE_SETUP ; ...space for file 1 on this unit
POPL R1 ; Have file 0...
MOVL R0,DISK_L_OTHER_PTR(R1) ; ...point to file 1...
MOVL R1,DISK_L_OTHER_PTR(R0) ; ...and file 1 point to file 0
BRW FOUND_IT ; Do the next UCB
```



```
0414 730 :  
0414 731 : Arrive here when we have the device configuration. In normal or loop forever  
0414 732 : mode, set a timer far enough in the future such that we can do a reasonable  
0414 733 : set of tests before the timer expires, but if our device gets hung, the  
0414 734 : program won't waste too much time before noticing. Let one-shot mode be a  
0414 735 : special case.  
0414 736 :  
0414 737 ALL_SET:  
0190'CF D5 0414 738 TSTL UNIT_LIST ; Anything to test?  
16 12 0418 739 BNEQ 10$ ; BR if yes  
012B'CF DF 041A 740 PUSHAL NOUNIT_SELECTED ; Else set up the error message...  
01 DD 041E 741 PUSHL #1 ; ...argument count...  
00741132 8F DD 0420 742 PUSHL #UETP$ _TEXT!ST$K_ERROR ; ...signal name...  
03 DD 0426 743 PUSHL #3 ; ...and parameter count  
0146'CF 14 DO 0428 744 MOVL #SS$ _BADPARAM,STATUS ; Set return status  
076C 31 042D 745 BRW ERROR_EXIT ; ...and give up, complaining  
0430 746 10$:  
0002'CF 04 A8 0430 747 BISW2 #SAFE_TO_UPDM,FLAG ; OK, safe to update UETINIDEV.DAT now  
0098'CF 0164'CF DO 0435 748 MOVL DEVNAM_LEN,DEVASC ; DEVASC will describe controller name  
13 0002'CF 04 E0 043C 749 BBS #ONE_SHOTV,FLAG,RESTART ; In one-shot mode start testing now...  
0442 750 ; ...else fall into TIME_IT  
0442 751  
0442 752 TIME_IT:  
0442 753 $SETIMR_S DAYTIM = THREEMIN,- ; Set timer AST to 3 minutes  
0442 754 ASTADR = TIME_OUT,-  
0442 755 EFN = #EFN2
```



```
0455 757 .SBTTL Test the Disk
0455 758
0455 759 RESTART:
0455 760 :
0455 761 : At this point the controller designation is in location DEV_NAME pointed to
0455 762 : by descriptor DEVDESC. The controller is known to be supported by this test.
0455 763 :
07 0002'CF 05 E1 0455 764 BBC #PATTERNV,FLAG,5$ ; BR if we're allowing various patterns
0166'CF 09A0'CF D0 0458 765 MOVL PATTERN_DATA,RANDOM1 ; We're forcing a specific pattern
0462 766 5$:
56 01A0'CF DE 0462 767 MOVAL RANDOM_DATA,R6 ; Set up...
57 00000200 8F D0 0467 768 MOVL #<WRITE_SIZE+3>/4,R7 ; ...a bufferful of longwords...
046E 769 10$:
07 0002'CF 05 E0 046E 770 BBS #PATTERNV,FLAG,15$ ; ...which can be either pattern...
0166'CF 016A'CF C0 0474 771 ADDL2 RANDOM2,RANDOM1 ; ...or pseudo_random...
047B 772 15$:
86 0166'CF D0 047B 773 MOVL RANDOM1,(R6)+ ; ...data...
EB 57 F5 0480 774 SOBGTR R7,10$ ; ...
0483 775 :
0483 776 : For each unit-file node in the queue, clean up any old version of the file we
0483 777 : want to create and create a new file. The FABs are already set up such that
0483 778 : each file uses PERCENTAGE blocks of the total space on the disk, the files
0483 779 : names are controller'unit'x, where 'x' is either '0' or '1', and the files
0483 780 : will be accessed in Block I/O mode by RMS. In normal and loop forever modes,
0483 781 : the test will proceed asynchronously with the file I/O. In one-shot mode,
0483 782 : I/O will be a single, synchronous write/read per file, handled separately.
0483 783 :
56 00000190'8F 0190'CF C1 0483 784 ADDL3 UNIT_LIST,#UNIT_LIST,R6 ; R6 will point to the current node
048D 785 20$:
OE A6 01 B0 048D 786 BSBW PRE CLEAN ; Remove leftover files from previous run
02 88 0490 787 MOVW #DISK_K CREATE,UETUNT$W_FUNC(R6) ; Indicate func in case error
OB A6 0494 788 BISB2 #UETUNT$M TESTABLE,- ; Assume file will be testable and...
OC 8A 0496 789 UETUNT$B FLAGS(R6)
OB A6 0498 790 BICB2 #DISK_M NOCREATE!DISK_M_NOEXTEND,-
048B 30 049A 791 UETUNT$B FLAGS(R6) ; ...$CREATE and $EXTEND succeed
01 E1 049C 792 $CREATE FAB = UETUNT$K_FAB(R6) ; Try to create a file on a disk unit
3D OB A6 04A7 793 BSBW CREATE CHECK ; Check for errors, recover from some
01BE C6 9A 04AA 794 BBC #UETUNT$V TESTABLE,- ; Skip $CONNECT if $CREATE failed
01B3 C6 04AC 795 UETUNT$B FLAGS(R6),40$
01BF C6 D0 04AF 796 MOVZBL DISK_C_NAM+NAM$B_RSL(R6),- ; Form an ASCID desc to the...
01B7 C6 04B3 797 DISK_Q_FILEDESC(R6)
OE A6 B4 04B6 798 MOVL DISK_C_NAM+NAM$B_RSA(R6),- ; ...file spec we actually used
0120 C6 D0 04BA 799 DISK_Q_FILEDESC+4(R6)
01AF C6 04BD 800 CLRW UETUNT$W_FUNC(R6) ; Reset func for error handling
0198 C6 D4 04C0 801 MOVL UETUNT$K_FAB+FAB$B_ALQ(R6),- ; Set current file size
10 A6 D4 04C4 802 DISK_L_SIZE(R6)
57 06F0'CF DE 04C7 803 CLRL UETUNT$K_RAB+RAB$B_BKT(R6) ; Initialize current block
02 0002'CF 04 E1 04CB 804 CLRL UETUNT$B_ITER(R6) ; Reset iteration count if loop forever
57 D4 04CE 805 MOVAL WRITE_NEXT,R7 ; Assume normal or loop mode asynch I/O
04D3 806 BBC #ONE_SHOTV,FLAG,30$ ; BR if that is indeed the case
04D9 807 CLRL R7 ; One-shot mode, no SUC= return
04DB 808 30$:
04DB 809 $CONNECT RAB = UETUNT$K_RAB(R6),- ; Set up the file's record access
04DB 810 ERR = RMS_ERROR,-
04DB 811 SUC = (R7)
04EC 812 40$:
56 66 C0 04EC 813 ADDL2 (R6),R6 ; Loop through all units in the queue
```


UETDISK00
V04-000

- VAX/VMS UETP DISK EXERCISER
Test the Disk

D 14

16-SEP-1984 01:22:47 VAX/VMS Macro V04-00
5-SEP-1984 04:24:57 [UETP.SRC]UETDISK00.MAR;1

Page 20
(10)

00000190'8F

56 D1 04EF 814
95 12 04F6 815

CMPL
BNEQ

R6,#UNIT_LIST
20\$

; Back at the head of the queue?
; BR if not


```
04F8 817 ;
04F8 818 ; All the $CREATEs and $CONNECTs are done. Before we try to hibernate in
04F8 819 ; normal or loop forever modes, or start I/O in one-shot mode, see if any
04F8 820 ; files were started successfully. R6 is still points to UNIT_LIST.
04F8 821 ;
04F8 822 50$:
04F8 823 ADDL2 (R6),R6 ; Point to the next possible node
00000190'8F 56 66 C0 04F8 824 CMPL R6,#UNIT_LIST ; Back at the head of the queue?
04F8 825 BNEQ 60$ ; Check this node if not
04F8 826 BRW TEST_OVER ; Nothing left to test so go home
00F0 03 12 0502 827 60$:
04F8 828 BBC #UETUNT$V TESTABLE,- ; Pass to next node if this one...
04F8 829 UETUNT$B_FLAGS(R6),50$ ; ...isn't testable
04F8 830 ; Otherwise start testing
04F8 831
04F8 832 BBS #ONE_SHOTV,FLAG,ONE_SHOT_TEST ; Synchronous I/O if one-shot mode
0C 0002'CF 04 E0 050C 833 ;
0512 834 ; We'll arrive here in normal or loop forever testing modes.
0512 835 ; All testable units have their files started up. Life proceeds asynchronously
0512 836 ; as write requests are issued and the results are verified. The AST routines
0512 837 ; are responsible for that as well as file expansion. The process terminates
0512 838 ; when the $SETIMR alarm goes off and wakes us from the following...
0512 839 ;
0512 840 $HIBER_S ; File testing continues via ASTs
0519 841 ;
0519 842 ; We return when all testing est finis. Clean up any messy files.
0519 843 ;
034B 30 0519 844 BSBW CLEAN_UP ; This closes and erases files
6D 11 051C 845 BRB SUC_EXIT ; Go finish off this pass of the test
```



```
051E 847 :  
051E 848 : Get here only in one-shot testing mode.  
051E 849 : All testable units have their files started up. We'll do one write/verify  
051E 850 : pair per file. Since everything is synchronous except error handling, all  
051E 851 : the code is inline.  
051E 852 :  
58 00000190'8F 0190'CF C1 051E 853 ONE_SHOT_TEST:  
051E 854 ADDL3 UNIT_LIST,#UNIT_LIST,R8 ; This will point to the current node  
0528 855 10$:  
0528 856 MOVAL UETUNT$K_RAB(R8),R6 ; Point to the RAB in current node  
E1 0528 857 BBC #UETUNT$V_TESTABLE,- ; Skip node if not testable  
052D 858 UETUNT$B_FLAGS(R8),20$  
052F 859 MOVL #1,RAB$L_BKT(R6) ; Pick a record to write  
28 A6 38 A6 01 DO 0532 859 MOVAL RANDOM_DATA,RAB$L_RBF(R6) ; Indicate write buffer  
01A0'CF DE 0536 860 $WRITE RAB = (R6),- ; Write a record  
053C 861 ERR = RMS_ERROR  
053C 862 BBC #UETUNT$V_TESTABLE,- ; Skip node if not testable  
01 0549 863 UETUNT$B_FLAGS(R8),20$  
0800 8F 00 29 0B A8 054B 864 MOVCS #0,(R8),#0,#WRITE_SIZE,DISK_K_BUFFER(R8) ; Clear the read buffer  
68 00 2C 054E 865  
031C C8 0555  
0558 866 $READ RAB = (R6),- ; Read a record  
0558 867 ERR = RMS_ERROR  
01A0'CF 0800 8F 29 0565 868 CMPC3 #READ_SIZE,RANDOM_DATA,- ; Verify a record  
031C C8 056C 869 DISK_K_BUFFER(R8)  
06 13 056F 870 BEQL 20$  
38 A6 D4 0571 871 CLRL RAB$L_BKT(R6) ; Indicate random data for last...  
0574 872 ; ...block read (see GET_PATTERN)  
0331 30 0574 873 BSBW VERIFY_ERROR ; Yell out the error  
0577 874 20$:  
58 68 C0 0577 875 ADDL2 (R8),R8 ; Loop through all units in the queue  
00000190'8F 58 D1 057A 876 CMPL R8,#UNIT_LIST ; Back at the head of the queue?  
A5 12 0581 877 BNEQ 10$ ; BR if more nodes to test  
0583 878  
0002'CF 02 A8 0583 879 BISW2 #TEST_OVERM,FLAG ; Indicate a normal test termination  
02DC 30 0588 880 BSBW CLEAN_UP ; Close and erase files  
0588 881 ;BRW SUC_EXIT ; Fall into finishing off the test
```



```
0146'CF  D5 058B 883 SUC_EXIT:
        6F 12 058B 884 TSTL STATUS ; Are we trying to exit?
        058F 885 BNEQ TEST_OVER STATUS ; BR if we are
        0591 886 STRNLOG_S LOGNAM = MODE,-
        0591 887 RSLLEN = BUFFER_PTR,-
        0591 888 RSLBUF = FAO_BUF ; Get the run mode
0014'CF 20 8A 05AA 889 BICB2 #LC_BITM,BUFFER ; Convert to upper case
0014'CF 4C 8F 91 05AF 890 CMPB #^A7L/,BUFFER ; Is this a loop for ever?
        40 12 05B5 891 BNEQ TEST_OVER ; BR if not
0002'CF 02 AA 05B7 892 BICW2 #TEST_OVERM,FLAG ; Reset the termination flag
0172'CF D6 05BC 893 INCL PASS ; Bump the pass count
        05C0 894 $FAO_S CTRSTR = PASS_MSG,-
        05C0 895 OUTLEN = BUFFER_PTR,-
        05C0 896 OUTBUF = FAO_BUF,-
        05C0 897 P1 = PASS,-
        05C0 898 P2 = ITERATION,-
        05C0 899 P3 = #0 ; Make the end of pass message
000C'CF DF 05DD 900 PUSHAL BUFFER_PTR ; Push the string desc.
        01 DD 05E1 901 PUSHL #1 ; Push arg count
00741133 8F DD 05E3 902 PUSHL #UETP$ TEXT!STSSK_INFO ; Push the signal name
00000000'GF 03 FB 05E9 903 CALLS #3,G^LIB$SIGNAL ; Print the end of pass message
        016E'CF D4 05F0 904 CLRL ITERATION ; Reset the iteration count
        FE4B 31 05F4 905 BRW TIME_IT ; Do the next pass
        05F7 906
        05F7 907 TEST_OVER:
0146'CF 10000001 8F D0 05F7 908 MOVL #SS$ _NORMAL!STSSM_INHIB_MSG,STATUS ; Set successful exit status
        0600 909 TEST_OVER STATUS:
        0600 910 $EXIT_S STATUS ; Exit with the status
```



```
060B 912 .SBTTL Setup Routine for One File on a Unit
060B 913 :++
060B 914 : FUNCTIONAL DESCRIPTION:
060B 915 : This routine is called to allocate space and do the general
060B 916 : initialization for the data structure associated with a file on a unit.
060B 917 :
060B 918 : CALLING SEQUENCE:
060B 919 : PUSHL number-of-this-file-on-this-unit
060B 920 : CALLS #1,UNIT_FILE_SETUP
060B 921 :
060B 922 : INPUT PARAMETERS:
060B 923 : A byte giving the cardinal number of the current file being set up on
060B 924 : the current unit (4(AP)).
060B 925 :
060B 926 : IMPLICIT INPUTS:
060B 927 : DEVDESC is a descriptor to the current device and unit name in DEV_NAME.
060B 928 : DIBBUF is the result of a $GETDEV on that device.
060B 929 : UNIT_NUMBER has the current unit number.
060B 930 : UNIT_LIST is the queue header to which a new node will be added.
060B 931 : DUMMY_FAB and DUMMY_RAB are adjacent and contain templates of a FAB and
060B 932 : a RAB, respectively.
060B 933 :
060B 934 : OUTPUT PARAMETERS:
060B 935 : R0 will have the address of the node just created.
060B 936 :
060B 937 : IMPLICIT OUTPUTS:
060B 938 : A new node will be added to the queue headed by UNIT_LIST.
060B 939 :
060B 940 : COMPLETION CODES:
060B 941 : NONE
060B 942 :
060B 943 : SIDE EFFECTS:
060B 944 : The space just allocated will be initialized.
060B 945 :
060B 946 : --
060B 947 UNIT_FILE_SETUP:
OFFC 060B 948 .WORD ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; Entry mask
060D 949
060D 950 $EXPREG_S PAGCNT = #PAGES,- ; Get a new node of demand zero memory
060D 951 RETADR = NEW_NODE
0190'CF 0198'DF 5D 061E 952 INSQTI @NEW_NODE,UNIT_LIST ; Put the new node in the unit list
56 0198'CF D0 0625 953 MOVL NEW_NODE,R6 ; Save a copy of its address
08 A6 01 90 062A 954 MOVB #1,DETUNTSB TYPE(R6) ; Set the structure type
031A 8F B0 062E 955 MOVW #UETUNTSC INDSIZ+DEVDEP_SIZE,-
09 A6 0632 956 UETUNT$W SIZE(R6) ; Set the structure size
0094 8F 28 0634 957 MOVC3 #FAB$C BFN+RAB$C BLN,-
0110 C6 0B24'CF 0638 958 DUMMY FAB,UETUNT$C FAB(R6) ; Save a FAB and a RAB away
57 0110 C6 DE 063E 959 MOVAL UETUNT$K_FAB(R6),R7 ; Save the FAB address
58 0160 C6 DE 0643 960 MOVAL UETUNT$K_RAB(R6),R8 ; Save the RAB address
3C A8 57 D0 0648 961 MOVL R7,RAB$K_FAB(R8) ; Set the FAB address in the RAB
24 A8 031C C6 DE 064C 962 MOVAL DISK_K_BUFFER(R6),RAB$K_UBF(R8) ; Set read buffer address...
28 A8 031C C6 DE 0652 963 MOVAL DISK_K_BUFFER(R6),RAB$K_RBF(R8) ; ...and write buffer address
```



```
0BB8'CF 0060 8F 28 0658 965      MOV C3 #NAM$C_BLN,DUMMY_NAM,- ; Set up a NAM block...
          01BB C6                DISK_C_NAM(R6)
          01BB C6 DE 0662 966      MOV AL DISK_C_NAM(R6),- ; ...so RMS knows exactly which file...
          0138 C6                UETUNT$C_FAB+FAB$S_L_NAM(R6)
          021B C6 DE 0669 969      MOV AL DISK_T_NAMFILSPC(R6),- ; ...to delete when cleaning up...
          01BF C6                DISK_C_NAM+NAM$S_L_RSA(R6); ...in case multiple tests are running
          59 C13E'CF 4E 0670 971    CVTLF DIBB0F7DIB$S_L_MAXBLOCK,R9 ; Use the total space on the disk...
          5A 59 04 45 0675 972    MULF3 #TOP_PCT,R9,R10 ; ...to get the max we'll use...
          59 CCCD3E4C 8F 44 0679 973  MULF2 #PERCENTAGE,R9 ; ...and how much each file is extended
          01AB C6 5A 4A 0680 974    CVTFL R10,DISK_L_TOP_PCT(R6) ; Store the max space for the unit...
          10 A7 59 4A 0685 975    CVTFL R9,FAB$S_L_ACQ(R7) ; ...and the original file size
          2C A7 14 A6 DE 0689 976
          59 0098'CF 01 A1 068E 977  MOV AL UETUNT$T_FILSPC(R6),FAB$S_L_FNA(R7); Set addr of filspec
          5A 59 B0 0694 979      ADDW3 #1,DEV DSC,R9 ; Device names are device'unit':
          27 5A B1 0697 980      MOVW R9,R10 ; NOTE: VMS guarantees length is OK
          5A 03 15 069A 981      CMPW R10,#FN_LEN ; Make file names as device'unit'x,...
          5A 27 B0 069C 982      BLEQ 10$ ; ...up to a maximum length
          069F 983 ; BR if length is OK
          069F 984 ; Otherwise force length to be OK
          069F 985 ; In this case, file number is lost...
          069F 986 ; ...but files may be distinguished...
          069F 987 10$ ; ...by their version number
          5B 04 AC 30 81 069F 988
          59 3A 00B7'CF 0098'CF 2C 06A4 989  ADDB3 #^A/0/,4(AP),R11 ; Form final filename char (file number)
          5A 5B 00B7'CF 0098'CF 2C 06AD 990  MOV C5 DEV DSC,DEV_NAME,#^A/://,R9,UETUNT$T_FILSPC(R6) ; Store device...
          63 50 0262'CF 98 06B9 991      MOVZBW DOTTST,R0 ; ...
          30 A7 0263'CF 50 28 06BE 992    MOV C3 R0,DOTTST+1,(R3) ; ...and extension
          024A'CF DE 06C4 993      MOV AL SYSTEST_DIR+1,FAB$S_L_DNA(R7) ; Set up typical default directory
          06CA 994 ; Note that MOV Cx thinks length is a word but RMS thinks it's a byte.
          34 A7 5A 59 81 06CA 995      ADDB3 R9,R10,FAB$B_FNS(R7) ; Store the file name string length...
          34 A7 0262'CF 80 06CF 997      ADDB2 DOTTST,FAB$B_FNS(R7) ; ... (dev:file.ext)
          35 A7 0249'CF 90 06D5 998      MOV B SYSTEST_DIR,FAB$B_DNS(R7) ; Store typical default directory length
          01A4 C6 0162'CF B0 06DB 999      MOVW UNIT_NUMBER,DISK_Q_UNIT(R6) ; Save the unit number...
          01A6 C6 04 AC 90 06E2 1000     MOV B 4(AP),DISK_B_FILE(R6) ; ...and the file number on this unit
          02 88 06E8 1001     BIS B2 #UETUNT$M_TESTABLE,- ; Assume file will be testable
          0B A6 06EA 1002     UETUNT$B_FLAGS(R6)
          50 56 D0 06EC 1003     MOVL R6,R0 ; Return node address as function value
          04 06EF 1004     RET
```



```
06F0 1006 .SBTTL Successful $READ AST Routine
06F0 1007 :++
06F0 1008 : FUNCTIONAL DESCRIPTION:
06F0 1009 : This routine will receive the AST when a $READ operation finishes for
06F0 1010 : a file on a unit. It verifies that the data read were correct and
06F0 1011 : starts up the next $WRITE. In situations detailed below, it will try
06F0 1012 : to extend the file.
06F0 1013 :
06F0 1014 : CALLING SEQUENCE:
06F0 1015 : Called via AST at I/O completion for all files on all units.
06F0 1016 :
06F0 1017 : INPUT PARAMETERS:
06F0 1018 : 4(AP) is RAB address
06F0 1019 :
06F0 1020 : IMPLICIT INPUTS:
06F0 1021 : All FABs and RABs.
06F0 1022 : An AST argument list with the ASTPRM value equal to the address of the
06F0 1023 : associated RAB.
06F0 1024 :
06F0 1025 : OUTPUT PARAMETERS:
06F0 1026 : NONE
06F0 1027 :
06F0 1028 : IMPLICIT OUTPUTS:
06F0 1029 : Fields may be modified in FABs and RABs.
06F0 1030 : If a file is being processed correctly and the test has not terminated,
06F0 1031 : a new block of the file is written.
06F0 1032 : A file may be extended.
06F0 1033 :
06F0 1034 : COMPLETION CODES:
06F0 1035 : NONE
06F0 1036 :
06F0 1037 : SIDE EFFECTS:
06F0 1038 : Other fields in the node associated with this file may be modified.
06F0 1039 : --
06F0 1040 WRITE_NEXT:
OFFC 06F0 1041 .WORD ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; Entry mask
06F2 1042
6D 09A5'CF DE 06F2 1043 MOVAL SSERROR,(FP) ; Declare exception handler
56 04 AC DO 06F7 1044 MOVL 4(AP),R6 ; Get the address of our RAB...
57 3C A6 DO 06FB 1045 MOVL RAB$L_FAB(R6),R7 ; ...and its associated FAB
58 FEAO C6 DE 06FF 1046 MOVAL -UETUNT$K_RAB(R6),R8 ; Set up a node pointer
0704 1047 ; These registers remain constant...
0704 1048 ; ...throughout this routine
```



```
0704 1050 :  
0704 1051 : Verify that the data read back were those that were written out. GET_PATTERN  
0704 1052 : below explains how we figure that out. Block 0 is an exception, however. To  
0704 1053 : RMS, block 0 would imply "write (or read) the next block of the file." We  
0704 1054 : therefore avoid its use in choosing a next block and additionally reserve  
0704 1055 : it to mean that the test has just started and there is no data to verify.  
0704 1056 :  
38 A6 D5 0704 1057 TSTL RAB$L_BKT(R6) ; If the last block accessed was 0...  
03 12 0707 1058 BNEQ 10$ ;  
0079 31 0709 1059 BRW DO_THE_WRITE ; ...then we have nothing to verify; BR  
06 0002'CF 05 E0 070C 1060 10$: BBS #PATTERNV_FLAG,15$ ; BR if we're forcing a pattern  
38 A6 03 D3 0712 1062 BITL #3,RAB$L_BKT(R6) ; Determine which pattern was used last  
0B 12 0716 1063 BNEQ 20$ ; BR if any bits are set  
28 B6 01A0'CF 0800 8F 29 0718 1064 15$: CMPC3 #READ_SIZE,RANDOM_DATA,@RAB$L_RBF(R6) ; Either ^B00 or pattern  
OC 11 0721 1066 BRB 30$ ; Rejoin common code  
28 B6 0800 8F 59 66 00BF 30 0723 1067 20$: BSBW GET_PATTERN ; Fill R9 with the pattern to verify  
00 2D 0726 1069 CMPC5 #0,(R6),R9,#READ_SIZE,@RAB$L_RBF(R6) ; Verify the buffer  
04 13 072F 1070 30$: BEQL CHECK_EXTEND ; BR if the strings were the same  
0174 30 0731 1072 BSBW VERIFY_ERROR ; Yell if strings differed (bad news)  
04 0734 1073 RET ; No more I/O to this file  
0735 1074 :  
0735 1075 : The data are verified. See if it's time to try to extend the file. File  
0735 1076 : extension is meant to test software as much as anything else. We will  
0735 1077 : attempt it if there is no flag contraindicating it for this file, we've done  
0735 1078 : a positive multiple of BAKERS $WRITES to the file and the total space across  
0735 1079 : all files on this unit doesn't exceed the value in DISK_L_TOP_PCT.  
0735 1080 : Most of the errors we will get from $EXTEND will be caused by exceeding disk  
0735 1081 : quota or filling up a pack. RMS_ERROR has code to ignore these two and we  
0735 1082 : must compensate here when FAB$L_ALQ is cleared by an $EXTEND failure.  
0735 1083 :  
0735 1084 : CHECK_EXTEND:  
03 E0 0735 1085 BBS #DISK_V_NOEXTEND,- ; BR if flag says don't try to extend  
4B 0B A8 0737 1086 UETUNT$B_FLAGS(R8),DO_THE_WRITE  
59 10 A8 D0 073A 1087 MOVL UETUNT$L_ITER(R8),R9 ; See if we've done...  
45 13 073E 1088 BEQL DO_THE_WRITE ; ...a positive...  
5A D4 0740 1089 CLRL R10 ; ... (EDIV wants quadword divisor)  
59 59 59 14 7B 0742 1090 EDIV #BAKERS,R9,R9,R9 ; ...multiple of BAKERS...  
59 D5 0747 1091 TSTL R9 ; ...writes to this file  
3A 12 0749 1092 BNEQ DO_THE_WRITE ; BR if there's a remainder  
59 01A7 C8 D0 074B 1093 MOVL DISK_L_OTHER_PTR(R8),R9 ; Calculate...  
01AF C9 C1 0750 1094 ADDL3 DISK_L_SIZE(R9),- ; ...the total space we're using...  
59 01AF C8 0754 1095 DISK_L_SIZE(R8),R9 ; ...on this unit  
01AB C8 59 D1 0758 1096 CMPL R9,DISK_L_TOP_PCT(R8) ; Have we used as much as we should?  
26 18 075D 1097 BGEQ DO_THE_WRITE ; BR if we've reached maximum space  
075F 1098 $EXTEND FAB = (R7),- ; We're OK, try to expand the file  
075F 1099 ERR = RMS_ERROR  
01AF C8 10 A7 C0 076C 1100 ADDL2 FAB$L_ALQ(R7),DISK_L_SIZE(R8) ; Compute new current size  
10 50 E8 0772 1101 BLBS RO,DO_THE_WRITE ; BR if we're OK  
08 88 0775 1102 BISB2 #DISK_M_NOEXTEND,- ; Prevent further attempts to extend  
0B A8 0777 1103 UETUNT$B_FLAGS(R8)  
59 10 A8 14 C7 0779 1104 DIVL3 #BAKERS,UETUNT$L_ITER(R8),R9 ; Number writes/writes-per-extend...  
10 A7 01AF C8 59 C7 077E 1105 DIVL3 R9,DISK_L_SIZE(R8),FAB$L_ALQ(R7) ; ...divides file size giving exten
```



```
0785 1107 :  
0785 1108 : Get the next block number to write.  
0785 1109 :  
0785 1110 DO_THE_WRITE:  
0785 1111 BBC #TEST_OVERV,FLAG,10$ ; BR if the test is still going  
04 078B 1112 RET ; Finished. Dismiss AST  
078C 1113 10$:  
078C 1114 ADDL2 RANDOM2,RANDOM1 ; Generate a "random" number  
59 0166'CF 016A'CF C0 078C 1115 BICL3 #^X80000000,RANDOM1,R9 ; From it get a new block number,...  
0166'CF 80000000 8F CB 0793 1116 CLRL R10 ; ... (EDIV wants quadword dividend)  
59 59 59 01AF C8 7B 079D 1117 EDIV DISK_L_SIZE(R8),R9,R9,R9 ; ...modulus the file size,...  
38 A6 59 D0 079F 1118 MOVL R9,RAB$L_BKT(R6) ; ...for next $WRITE  
E0 13 07AA 1119 BEQL 10$ ; But don't allow 0 as a block number!  
07AC 1120 :  
07AC 1121 : We have the block number to write next; now we must determine what data will  
07AC 1122 : be written.  
07AC 1123 :  
06 0002'CF 05 E0 07AC 1124 BBS #PATTERNV,FLAG,15$ ; BR if we've forced pattern data  
38 A6 03 D3 07B2 1125 BITL #3,RAB$L_BKT(R6) ; Determine which pattern to write next  
0B 12 07B6 1126 BNEQ 20$ ; BR if any bits are set  
07B8 1127 15$:  
28 B6 0800 8F 28 07B8 1128 MOVC3 #WRITE_SIZE,- ; Either random or forced pattern  
01A0'CF 0C 11 07BC 1129 RANDOM_DATA,@RAB$L_RBF(R6)  
07C1 1130 BRB 30$ ; Go write the block  
07C3 1131 20$:  
28 B6 0800 8F 59 66 00 30 07C3 1132 BSBW GET_PATTERN ; Fill R9 with the pattern to write  
2C 07C6 1133 MOVC5 #0,(R6),R9,#WRITE_SIZE,@RAB$L_RBF(R6) ; Buffer gets pattern data  
07CF 1134 30$:  
10 A8 D6 07CF 1135 INCL UETUNT$L_ITER(R8) ; Count this file's writes  
07D2 1136 $WRITE RAB = @4(AP),- ; Write another block to current file  
07D2 1137 ERR = RMS_ERROR,-  
07D2 1138 SUC = READ_NEXT  
04 07E4 1139 RET ; Dismiss the AST  
07E5 1140  
07E5 1141  
07E5 1142  
07E5 1143  
07E5 1144 :  
07E5 1145 : Subroutine to fill the low order byte of R9 with the bit pattern of bits 0  
07E5 1146 : and 1 of the block number (presumably random) in RAB$L_BKT. The pattern will  
07E5 1147 : be carried throughout the buffer on a $WRITE and later used to verify that  
07E5 1148 : the data were written out and read in correctly. This subroutine will not  
07E5 1149 : be used if the bit pattern is ^B00 or if the flag PATTERNM is set; in those  
07E5 1150 : cases the contents of the RANDOM_DATA buffer will be used.  
07E5 1151 :  
59 38 A6 02 00 EE 07E5 1152 GET_PATTERN:  
05 19 07EB 1153 EXTIV #0,#2,RAB$L_BKT(R6),R9 ; Get sign-extended bit pattern  
59 55 8F 90 07ED 1154 BLSS 10$ ; BR if ^B11 or ^B10 pattern  
05 07F1 1155 MOVB #^B01010101,R9 ; Prepare to propagate the bit pattern  
07F2 1156 RSB  
07F2 1157 10$:  
59 04 59 E8 07F2 1158 BLBS R9,20$ ; BR if ^B11 pattern (R9 will have -1)  
59 AA 8F 90 07F5 1159 MOVB #^B10101010,R9 ; Prepare to propagate the bit pattern  
05 07F9 1160 20$:  
07F9 1161 RSB
```



```
07FA 1163 .SBTTL Successful $WRITE AST Routine
07FA 1164 :++
07FA 1165 : FUNCTIONAL DESCRIPTION:
07FA 1166 : This routine will receive the AST when a $WRITE operation finishes for
07FA 1167 : a file on a unit. It starts up a $READ operation and dismisses the
07FA 1168 : AST.
07FA 1169 :
07FA 1170 : CALLING SEQUENCE:
07FA 1171 : Called via AST at I/O completion for any file on any unit.
07FA 1172 :
07FA 1173 : INPUT PARAMETERS:
07FA 1174 : 4(AP) is RAB address
07FA 1175 :
07FA 1176 : IMPLICIT INPUTS:
07FA 1177 : ALL RABs (and FABs, implicitly).
07FA 1178 : An AST argument list with the ASTPRM value equal to the address of the
07FA 1179 : associated RAB.
07FA 1180 :
07FA 1181 : OUTPUT PARAMETERS:
07FA 1182 : NONE
07FA 1183 :
07FA 1184 : IMPLICIT OUTPUTS:
07FA 1185 : Fields may be modified in RABs.
07FA 1186 : If a file is being processed correctly and the test has not terminated,
07FA 1187 : the block of the file just written is read back in as verification.
07FA 1188 :
07FA 1189 : COMPLETION CODES:
07FA 1190 : NONE
07FA 1191 :
07FA 1192 : SIDE EFFECTS:
07FA 1193 : NONE
07FA 1194 :
07FA 1195 :--
07FA 1196 :
07FA 1197 READ_NEXT:
OFFC 07FA 1198 .WORD ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; Entry mask
07FC 1199
07FC 1200 MOVAL SSERROR,(FP) ; Declare exception handler
0801 1201 BBS #TEST_OVERV,FLAG,10$ ; Stop ASAP if test is over
0807 1202 MOVL 4(AP),R6 ; Get the address of our RAB
28 B6 0808 1203 MOVCS #0,(R6),#0,#WRITE_SIZE,@RAB$RBF(R6) ; Clear the read buffer
0814 1204 $READ RAB = @4(AP),- ; Read back the block just written
0814 1205 ERR = RMS_ERROR,-
0814 1206 SUC = WRITE_NEXT
0826 1207 10$:
04 0826 1208 RET ; Dismiss the AST
```



```
0827 1210 .SBTTL Left Over File Cleanup Routine
0827 1211 :++
0827 1212 : FUNCTIONAL DESCRIPTION:
0827 1213 : If there are files left over from a previous run of the disk test (as
0827 1214 : would happen, for example, if the system crashed during a run), this
0827 1215 : routine will clean up the highest version of the file we're about to
0827 1216 : create. This is preferred over a supersede operation because supersede
0827 1217 : will not delete the old file's space until after the new file is
0827 1218 : created.
0827 1219 :
0827 1220 : CALLING SEQUENCE:
0827 1221 : BSBW PRE_CLEAN
0827 1222 :
0827 1223 : INPUT PARAMETERS:
0827 1224 : NONE
0827 1225 :
0827 1226 : IMPLICIT INPUTS:
0827 1227 : Current FAB, pointed to as UETUNT$K_FAB(R6).
0827 1228 :
0827 1229 : OUTPUT PARAMETERS:
0827 1230 : NONE
0827 1231 :
0827 1232 : IMPLICIT OUTPUTS:
0827 1233 : NONE
0827 1234 :
0827 1235 : COMPLETION CODES:
0827 1236 : NONE
0827 1237 :
0827 1238 : SIDE EFFECTS:
0827 1239 : File may be deleted. Note that we do not check for any errors. An
0827 1240 : error from the $ERASE may be expected in situations where this test
0827 1241 : is run in parallel with itself on a given disk (e.g., from separate
0827 1242 : nodes in a cluster to an MSCP disk).
0827 1243 : R0 is trashed.
0827 1244 :
0827 1245 :--
0827 1246 :
0827 1247 PRE_CLEAN:
0827 1248 PUSHR #*M<R1,R2,R3,R4,R5,R6,R7,R8,R9,R10> ; We clobber nothing here...
0827 1249 MOVL SP,R10 ; ...including our stack!
0827 1250 MOVAL UETUNT$K_FAB(R6),R7 ; Use R7 for more visible access
0827 1251 MOVZBL FAB$B_BLN(R7),R8 ; Again, we clobber nothing...
0827 1252 SUBL2 R8,SP- ; ...so we'll do our RMS access...
0827 1253 MOVCL R8,(R7),(SP) ; ...using a FAB on the stack
0827 1254 MOVL SP,R9 ; Save a pointer to our new FAB
0827 1255 $OPEN FAB = R9 ; See if old version of file is there
0827 1256 BLBC R0,10$ ; We can do nothing if any error
0827 1257 $CLOSE FAB = R9 ; We found one...
0827 1258 $ERASE FAB = R9 ; ...get rid of it
0827 1259 10$:
0827 1260 MOVL R10,SP ; Restore our stack...
0827 1261 POPR #*M<R1,R2,R3,R4,R5,R6,R7,R8,R9,R10> ; ...and all our registers
0827 1262 RSB
```

07FE 8F BB 0827 1248
5A 5E DO 0827 1249
57 0110 C6 DE 0827 1250
58 01 A7 9A 0833 1251
5E 58 C2 0837 1252
6E 67 58 28 083A 1253
59 5E DO 083E 1254
12 50 E9 0841 1255
084A 1256
084D 1257
0856 1258
085F 1259
5E 5A DO 085F 1260
07FE 8F BA 0862 1261
05 0866 1262


```
0867 1264 .SBTTL File Cleanup Routine
0867 1265 :++
0867 1266 : FUNCTIONAL DESCRIPTION:
0867 1267 :   This routine will clean up the files used for the test.
0867 1268 :
0867 1269 : CALLING SEQUENCE:
0867 1270 :   BSBW   CLEAN_UP
0867 1271 :
0867 1272 : INPUT PARAMETERS:
0867 1273 :   NONE
0867 1274 :
0867 1275 : IMPLICIT INPUTS:
0867 1276 :   ALL FABs. UNIT_LIST points to a queue with unit-file nodes.
0867 1277 :
0867 1278 : OUTPUT PARAMETERS:
0867 1279 :   NONE
0867 1280 :
0867 1281 : IMPLICIT OUTPUTS:
0867 1282 :   NONE
0867 1283 :
0867 1284 : COMPLETION CODES:
0867 1285 :   NONE
0867 1286 :
0867 1287 : SIDE EFFECTS:
0867 1288 :   Files are closed, deleted and erased. All registers may be trash.
0867 1289 :
0867 1290 :--
0867 1291
0867 1292 CLEAN_UP:
0867 1293   MOVAL   UNIT_LIST,R6           ; This will point to the current unit
086C 1294 10$:
086C 1295   ADDL2   (R6),R6               ; Loop through all nodes
086F 1296   CMPL   R6,#UNIT_LIST       ; Back at the head of the queue yet?
0876 1297   BEQL   40$              ; BR if we've finished
0878 1298
0878 1299   BBS     #DISK_V NOCREATE,-    ; Don't try $CLOSE if $CREATE failed
087A 1300   UETUNT$B_FLAGS(R6),10$
087D 1301   MOVAL   UETUNT$K_FAB(R6),R8 ; Get FAB address
0882 1302 20$:
0882 1303   $WAIT   RAB = UETUNT$K_RAB(R6) ; Twiddle thumbs until I/O completes
088D 1304   $CLOSE  FAB = (R8)         ; Try to close the file
0896 1305   ADDL2   UETUNT$L_ITER(R6),ITERATION ; Count total I/O for test
089C 1306   $ERASE  FAB = (R8)         ; File closed, remove directory entry
08A5 1307   BRB     10$
08A7 1308 40$:
08A7 1309   RSB                      ; We're all cleaned up
```



```
08A8 1311 .SBTTL Bum Data Routine
08A8 1312 :++
08A8 1313 : FUNCTIONAL DESCRIPTION:
08A8 1314 :   Print an error message if we get a mismatch during data verification.
08A8 1315 :
08A8 1316 : CALLING SEQUENCE:
08A8 1317 :   CMPCx   ...@RAB$L_RBF(R6)
08A8 1318 :   BEQL    data-was-good
08A8 1319 :   BSBW    VERIFY_ERROR
08A8 1320 :
08A8 1321 : INPUT PARAMETERS:
08A8 1322 :   NONE
08A8 1323 :
08A8 1324 : IMPLICIT INPUTS:
08A8 1325 :   R1 points to the good data if we just read random data or forced
08A8 1326 :   pattern data, never-never land if we read generated pattern data.
08A8 1327 :   R3 points to the offending byte of differing data.
08A8 1328 :   R6 points to the RAB for the bad file.
08A8 1329 :   R8 points to the unit data block.
08A8 1330 :   R9 will have the pattern if we matched generated pattern data.
08A8 1331 :
08A8 1332 : OUTPUT PARAMETERS:
08A8 1333 :   NONE
08A8 1334 :
08A8 1335 : IMPLICIT OUTPUTS:
08A8 1336 :   Message to SYS$OUTPUT and SYS$ERROR.
08A8 1337 :   Registers R2-R8 and R10-R11 returned unscathed.
08A8 1338 :
08A8 1339 : COMPLETION CODES:
08A8 1340 :   NONE
08A8 1341 :
08A8 1342 : SIDE EFFECTS:
08A8 1343 :   NONE
08A8 1344 :
08A8 1345 :--
08A8 1346
08A8 1347 VERIFY_ERROR:
06 0002'CF 05 E0 08A8 1348 BBS #PATTERNV,FLAG,5$ ; BR if we used forced pattern data
   38 A6 03 D3 08AE 1349 BITL #3,RAB$L_BKT(R6) ; Did we have random data?
   03 12 08B2 1350 BNEQ 10$ ; BR if we used generated pattern data
   59 61 90 08B4 1351 5$: MOV B (R1),R9 ; Copy the byte where failure was found
   7E 63 9A 08B7 1352 10$: MOV B (R3),-(SP) ; Save the bad...
   7E 59 9A 08BA 1353 MOV B R9,-(SP) ; ...the good...
7E 53 28 A6 C3 08BD 1354 SUBL3 RAB$L_RBF(R6),R3,-(SP) ; ...the place...
   01B3 C8 7F 08C2 1355 PUSH AQ DISK & FILEDESC(R8) ; ...the file spec...
   000F0004 8F DD 08C6 1356 PUSHL #^XF0004 ;
   0074801A 8F DD 08CC 1357 PUSHL #UETP$ DATADEVERR!STSSK_ERROR ; ...and the ugly info
   0142'CF D6 08D2 1358 INCL ERROR_COUNT ; Keep a running error count
   0142'CF DD 08D6 1359 PUSHL ERROR_COUNT ;
   00A0'CF DF 08DA 1360 PUSHAL PROCESS_NAME ;
   00010002 8F DD 08DE 1361 PUSHL #^X10002 ;
   00748022 8F DD 08E4 1362 PUSHL #UETP$ ERBOXPROC!STSSK_ERROR ; Have error stand out in log file
00000000'GF 0A FB 08EA 1363 CALLS #10,G^CIB$SIGNAL ; Let the user know what happened
   02 8A 08F1 1364 BICB2 #UETUNT$M TESTABLE,- ; Indicate that this file is no good
   FEAB C6 08F3 1365 UETUNT$B_FLAGS-UETUNT$K_RAB(R6)
```


UETDISK00
V04-000

- VAX/VMS UETP DISK EXERCISER
Bum Data Routine

D 15

16-SEP-1984 01:22:47 VAX/VMS Macro V04-00
5-SEP-1984 04:24:57 [UETP.SRC]UETDISK00.MAR;1

Page 33
(22)

0000 31 08F6 1368

BRW

DESTP_CHECK

; Exit via possible additional message


```
08F9 1370 .SBTTL Check if We've Stopped Testing a Unit
08F9 1371 :++
08F9 1372 : FUNCTIONAL DESCRIPTION:
08F9 1373 : In normal and loop forever modes, we warn the user that a device is
08F9 1374 : seen to be totally untestable (at least during this pass).
08F9 1375 :
08F9 1376 : CALLING SEQUENCE:
08F9 1377 : BSBW DESTP_CHECK
08F9 1378 :
08F9 1379 : INPUT PARAMETERS:
08F9 1380 : NONE
08F9 1381 :
08F9 1382 : IMPLICIT INPUTS:
08F9 1383 : R8 points to the unit data block
08F9 1384 :
08F9 1385 : OUTPUT PARAMETERS:
08F9 1386 : NONE
08F9 1387 :
08F9 1388 : IMPLICIT OUTPUTS:
08F9 1389 : NONE
08F9 1390 :
08F9 1391 : COMPLETION CODES:
08F9 1392 : NONE
08F9 1393 :
08F9 1394 : SIDE EFFECTS:
08F9 1395 : Possible error message if all files for this unit are marked as
08F9 1396 : untestable.
08F9 1397 :
08F9 1398 :--
08F9 1399 :
08F9 1400 DESTP_CHECK:
30 0002'CF 56 DD 08F9 1401 PUSHL R6 ; We'll affect no registers herein
56 04 EO 08FB 1402 BBS #ONE_SHOTV,FLAG,20$ ; This is meaningless in one-shot mode
56 58 DO 0901 1403 MOVL R8,R6 ; Set up a working pointer to node
0904 1404 10$:
56 01 EO 0904 1405 BBS #UETUNT$V TESTABLE,- ; If this file is still good...
28 0B A6 0906 1406 UETUNT$B FLAGS(R6),20$ ; ...then we need no message
56 01A7 C6 DO 0909 1407 MOVL DISK_L_OTHER_PTR(R6),R6 ; Point to next node for this unit
58 56 D1 090E 1408 CMPL R6,R8 ; Back where we started?
F1 12 0911 1409 BNEQ 10$ ; BR if not - check this one, too
00 DD 0913 1410 PUSHL #0 ; None testable - we need message
7E 01A4 C8 3C 0915 1411 MOVZWL DISK_W_UNIT(R8),-(SP) ; Unit number
0098'CF DF 091A 1412 PUSHAL DEV$SC ; Controller
00A0'CF DF 091E 1413 PUSHAL PROCESS_NAME ; Image name
04 DD 0922 1414 PUSHL #4
007481A2 8F DD 0924 1415 PUSHL #UETP$ DESTP!STSSK_ERROR
00000000'GF 06 FB 092A 1416 CALLS #6,G^LIB$SIGNAL
0931 1417 20$:
56 8ED0 0931 1418 POPL R6
05 0934 1419 RSB
```



```
0935 1421 .SBTTL $CREATE-Specific Error Checker
0935 1422 :++
0935 1423 : FUNCTIONAL DESCRIPTION:
0935 1424 : This routine handles error returns from $CREATEs of test files. It
0935 1425 : allows us to first try creating a file in device:[SYSTEST], and then
0935 1426 : for system packs with multiple system directory hierarchies, to try
0935 1427 : creating the file in device:[SYS0.SYSTEST]. This means we can test
0935 1428 : all packs, whether mounted as system packs or not. The routine also
0935 1429 : allows us to recover from most RMS$_FUL errors by trying to allocate
0935 1430 : a file that is only 1% of the total space on a pack.
0935 1431 :
0935 1432 : CALLING SEQUENCE:
0935 1433 : BSBW CREATE_CHECK
0935 1434 :
0935 1435 : INPUT PARAMETERS:
0935 1436 : NONE
0935 1437 :
0935 1438 : IMPLICIT INPUTS:
0935 1439 : R6 points to our unit data block for this file on this device.
0935 1440 : The FAB associated with the RMS $CREATE as UETUNT$K_FAB(R6).
0935 1441 :
0935 1442 : OUTPUT PARAMETERS:
0935 1443 : NONE
0935 1444 :
0935 1445 : IMPLICIT OUTPUTS:
0935 1446 : Error message
0935 1447 :
0935 1448 : COMPLETION CODES:
0935 1449 : NONE
0935 1450 :
0935 1451 : SIDE EFFECTS:
0935 1452 : If the error can be corrected by retrying with a different directory,
0935 1453 : the test file will be created. If the file can be created with a
0935 1454 : smaller allocation, it will be. Otherwise, the program may exit,
0935 1455 : depending on severity of the error. If the program does not exit
0935 1456 : and the file is not created, DISK_M_NOCREATE is set in UETUNT$B_FLAGS
0935 1457 : by the RMS error handler. R0 and R2 are trashed.
0935 1458 :
0935 1459 :--
0935 1460 :
0935 1461 : CREATE_CHECK:
0935 1462 : BLBS R0,30$ ; BR if there was no error
0935 1463 : MOVAL UETUNT$K_FAB(R6),R2 ; Point to our FAB for convenience
0935 1464 : CMPL #RMS$_DNF,FAB$L_STS(R2) ; Did we get directory not found error?
0935 1465 : BNEQ 10$ ; BR if not
0935 1466 : MOVB SYS0 SYSTEST DIR,- ; Yes, try it again...
0935 1467 : FAB$B DNS(R2)
0935 1468 : MOVAL SYS0 SYSTEST DIR+1,- ; ...using...
0935 1469 : FAB$C DNA(R2) ; ...V3 system directory hierarchies
0935 1470 : BSBW PRE_CLEAN ; Try again to clean up old file
0935 1471 : $CREATE FAB=(R2) ; Try again to create new file
0935 1472 : BLBS R0,30$ ; BR if we got it this time
0935 1473 :
0935 1474 : 10$:
0935 1475 : CMPL #RMS$_FUL,R0 ; Did we fail because file was too big?
0935 1476 : BNEQ 20$ ; BR if not
0935 1477 : MULF3 #^F100.0,#PERCENTAGE,R0 ; Try to get...
```

```
59 50 E8
52 0110 C6 DE
08 A2 00000000'8F D1
1B 12
0253'CF 90
35 A2 094B
0254'CF DE
30 A2 0951
FED1 30
2F 50 E8
```

```
50 00000000'8F D1
1F 12
50 CCCD3E4C 8F 000043C8 8F 45
```


UETDISK00
V04-000

- VAX/VMS UETP DISK EXERCISER
\$CREATE-Specific Error Checker

G 15

16-SEP-1984 01:22:47 VAX/VMS Macro V04-00
5-SEP-1984 04:24:57 [UETP.SRC]UETDISK00.MAR;1

Page 36
(24)

50	50	4A	0977	1478	CVTFL	R0,R0		
10 A2	50	C6	097A	1479	DIVL2	R0,FAB\$L_ALQ(R2)		; ...1% of this unit's space...
			097E	1480	\$CREATE	FAB = (R2)		; ...and try again for the file
	07 50	E8	0987	1481	BLBS	R0,30\$		; BR if we got the file
			098A	1482				
			098A	1483				
	52	DD	098A	1484	PUSHL	R2		; We don't know which error we got...
0A88'CF	01	FB	098C	1485	CALLS	#1,RMS_ERROR		; ...so let real routine handle it
			0991	1486				
		05	0991	1487	RSB			


```
0992 1489 .SBTTL Timer Expiration Routine
0992 1490 :++
0992 1491 : FUNCTIONAL DESCRIPTION:
0992 1492 :   This routine will signal the end of the test when we time out.
0992 1493 :
0992 1494 : CALLING SEQUENCE:
0992 1495 :   Called via AST at $SETIMR expiration.
0992 1496 :
0992 1497 : INPUT PARAMETERS:
0992 1498 :   NONE
0992 1499 :
0992 1500 : IMPLICIT INPUTS:
0992 1501 :   NONE
0992 1502 :
0992 1503 : OUTPUT PARAMETERS:
0992 1504 :   NONE
0992 1505 :
0992 1506 : IMPLICIT OUTPUTS:
0992 1507 :   NONE
0992 1508 :
0992 1509 : COMPLETION CODES:
0992 1510 :   Sets a flag to indicate timer expiration.
0992 1511 :
0992 1512 : SIDE EFFECTS:
0992 1513 :   The TEST_OVER flag is set in FLAG. This will tell the AST level
0992 1514 :   routines to not schedule any further I/O.
0992 1515 :   The program is aWAKEd from its HIBERnating state.
0992 1516 :   Note that no cleanup is done here.
0992 1517 :
0992 1518 :--
0992 1519 :
0992 1520 TIME_OUT:
0002'CF 02 04 0992 1521 .WORD ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; Entry mask
0994 1522
0994 1523 BISW2 #TEST_OVERM,FLAG ; Tell AST routines to stop
0999 1524 $WAKE_S ; Resume normal execution
09A4 1525 RET
```



```
09A5 1527 .SBTTL System Service Exception Handler
09A5 1528 :++
09A5 1529 : FUNCTIONAL DESCRIPTION:
09A5 1530 : This routine is executed if a software or hardware exception occurs or
09A5 1531 : if a LIB$SIGNAL system service is used to output a message.
09A5 1532 :
09A5 1533 : CALLING SEQUENCE:
09A5 1534 : Entered via an exception from the system
09A5 1535 :
09A5 1536 : INPUT PARAMETERS:
09A5 1537 : ERROR_COUNT = previous cumulative error count
09A5 1538 :
09A5 1539 : AP ---->
09A5 1540 :
09A5 1541 : SIGNAL ARY PNT
09A5 1542 : MECH ARY PNT
09A5 1543 :
09A5 1544 :
09A5 1545 : 4
09A5 1546 :
09A5 1547 : ESTABLISH FP
09A5 1548 :
09A5 1549 : DEPTH
09A5 1550 :
09A5 1551 : R0
09A5 1552 :
09A5 1553 : R1
09A5 1554 :
09A5 1555 : N
09A5 1556 :
09A5 1557 : CONDITION NAME
09A5 1558 :
09A5 1559 : N-3 ADDITIONAL
09A5 1560 : LONG WORD ARGS
09A5 1561 :
09A5 1562 : PC
09A5 1563 :
09A5 1564 : PSL
09A5 1565 :
09A5 1566 : IMPLICIT INPUTS:
09A5 1567 : NONE
09A5 1568 :
09A5 1569 : OUTPUT PARAMETERS:
09A5 1570 : NONE
09A5 1571 :
09A5 1572 : IMPLICIT OUTPUTS:
09A5 1573 : NONE
09A5 1574 :
09A5 1575 : COMPLETION CODES:
09A5 1576 : SSS_NORMAL if it's a UETP condition or RMS error.
09A5 1577 : Error status from exception, otherwise.
09A5 1578 :
09A5 1579 : SIDE EFFECTS:
09A5 1580 : May branch to ERROR_EXIT.
09A5 1581 : May print a message.
09A5 1582 : --
09A5 1583 :
```

2	
SIGNAL ARY PNT	
MECH ARY PNT	
4	
ESTABLISH FP	
DEPTH	Mechanism Array
R0	
R1	
N	
CONDITION NAME	
N-3 ADDITIONAL LONG WORD ARGS	Signal Array
PC	
PSL	


```

                                09A5 1584 SSERROR:
OFFC 09A5 1585 .WORD ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; Entry mask
      09A7 1586
      09A7 1587 $SETAST_S ENBFLG = #0 ; Disable AST delivery
50 01 DD 09B0 1588 PUSHL #1 ; Assume ASTs were enabled
    09 D1 09B2 1589 CMPL S^#SS$_WASSET,R0 ; Were ASTs enabled?
    02 13 09B5 1590 BEQL 10$ ; BR if they were
    6E D4 09B7 1591 CLRL (SP) ; Set ASTs to remain disabled
      09B9 1592 10$:
      09B9 1593 $SETSFM_S ENBFLG = #0 ; Disable SS failure mode
50 01 DD 09C2 1594 PUSHL #1 ; Assume SS failure mode was enabled
    09 D1 09C4 1595 CMPL S^#SS$_WASSET,R0 ; Was SS failure mode enabled?
    02 13 09C7 1596 BEQL 20$ ; BR if it was
    6E D4 09C9 1597 CLRL (SP) ; Set SS failure mode to remain off
      09CB 1598 20$:
56 04 AC D0 09CB 1599 MOVL CHF$_SIGARGLST(AP),R6 ; Get the signal array pointer
59 04 A6 7D 09CF 1600 MOVQ CHF$_SIG_NAME(R6),R9 ; Get NAME in R9 and ARG1 in R10
      10 ED 09D3 1601 CMPZV #ST$_FAC_NO,- ; Is this a message from LIB$SIGNAL?
      0C 09D5 1602 #ST$_FAC_NO,-
      59 09D6 1603 R9,#UTPS_FACILITY
      14 12 09DC 1604 BNEQ 30$ ; BR if this is not a UETP exception
      66 02 C2 09DE 1605 SUBL2 #2,CHF$_SIG_ARGS(R6) ; Drop the PC and PSL
      21 11 09E1 1606 $PUTMSG_S MSGVEC = CHF$_SIG_ARGS(R6) ; Print the message
      09F0 1607 BRB 40$ ; Restore ASTs and SS fail mode
59 0000045C 8F D1 09F2 1608 30$:
      32 12 09F9 1609 CMPL #SS$_SSFAIL,R9 ; RMS failures are SysSvc failures
      10 ED 09FB 1610 BNEQ 50$ ; BR if this can't be an RMS failure
      0C 09FD 1611 CMPZV #ST$_FAC_NO,- ; Is it an RMS failure?
      01 5A 09FE 1612 #ST$_FAC_NO,-
      2B 12 0A00 1613 R10,#RMS_FACILITY
5A F0000000 8F CA 0A02 1614 BNEQ 50$ ; BR if not
      08 A6 04 39 0A09 1615 BICL2 #^XF0000000,R10 ; Strip control bits from status code
      14 0A0D 1616 MATCHC #4,CHF$_SIG_ARG1(R6),- ; Is it an RMS failure for which...
      004D'CF 0A0E 1617 #NRAT_LENGTH,-
      1A 13 0A11 1618 NO RMS_AST_TABLE
      0A13 1619 BEQL 50$ ; ...no AST can be delivered?
      01 BA 0A13 1620 40$: ; BR if so - must give error here
      0A15 1621 POPR #^M<R0> ; Restore SS failure mode...
      01 BA 0A1E 1622 $SETSFM_S ENBFLG = R0 ;
      0A20 1623 POPR #^M<R0> ; Restore AST enable...
      50 01 D0 0A29 1624 $SETAST_S ENBFLG = R0 ;
      04 0A2C 1625 MOVL S^#SS$_NORMAL,R0 ; Supply a standard status for exit
      0A2D 1626 RET ; Resume processing (or goto RMS_ERROR)
      0A2D 1627 50$:
0146'CF 59 D0 0A2D 1628 MOVL R9,STATUS ; Save the status
      58 D4 0A32 1629 CLRL R8 ; Assume for now it's not SS failure
59 0000045C 8F D1 0A34 1630 CMPL #SS$_SSFAIL,R9 ; But is it a System Service failure?
      38 12 0A3B 1631 BNEQ 70$ ; BR if not - no special case message
      0A3D 1632 $GETMSG_S MSGID = R10,- ; Get SS failure code associated text
      0A3D 1633 MSGLEN = BUFFER_PTR,-
      0A3D 1634 BUFADR = FAO_BUF,-
      0A3D 1635 FLAGS = #14,-
      0A3D 1636 OUTADR = MSG_BLOCK
0177'CF 95 0A54 1637 TSTB MSG_BLOCK+1 ; Get FAO arg count for SS failure code
      16 13 0A58 1638 BEQL 60$ ; Don't use $GETMSG if no $FAO args...
000C'CF DF 0A5A 1639 PUSHAL BUFFER_PTR ; ...else build up...
      01 DD 0A5E 1640 PUSHL #1 ; ...a message describing...

```



```
00741130 8F DD 0A60 1641 PUSH  #UETPS TEXT ; ...why the System Service failed
          00 5A FO 0A66 1642 INSV  R10,#STSSV SEVERITY,- ; Give the message...
          6E 03 DD 0A69 1643 ; ...the correct severity code
          58 03 DO 0A6B 1644 MOVL  #3,R8 ; Count the number of args we pushed
          05 11 DD 0A6E 1645 BRB   70$
          5A DD 0A70 1646 60$:
          58 01 DO 0A70 1647 PUSH  R10 ; Save SS failure code
          01 DO 0A72 1648 MOVL  #1,R8 ; Count the number of args we pushed
          04 C5 0A75 1649 70$:
          57 66 04 C5 0A75 1650 MULL3 #4,CHF$L_SIG_ARGS(R6),R7 ; Convert longwords to bytes
          5E 57 C2 0A79 1651 SUBL2 R7,SP ; Save the current signal array...
          6E 04 A6 57 28 0A7C 1652 MOVC3 R7,CHF$L_SIG_NAME(R6),(SP) ; ...on the stack
          7E 66 58 C1 0A81 1653 ADDL3 R8,CHF$L_SIG_ARGS(R6),-(SP) ; Push the current arg count
          0114 31 0A85 1654 BRW   ERROR_EXIT
          0A88 1655
```



```

OA88 1657 .SBTTL General RMS Error Handler
OA88 1658 :++
OA88 1659 : FUNCTIONAL DESCRIPTION:
OA88 1660 : This routine handles error returns from RMS calls.
OA88 1661 :
OA88 1662 : CALLING SEQUENCE:
OA88 1663 : Called by RMS when a file processing error is found. Will also be
OA88 1664 : called by TRY SYS0 if an error other than directory-not-found is
OA88 1665 : encountered when creating a test file.
OA88 1666 :
OA88 1667 : INPUT PARAMETERS:
OA88 1668 : The FAB or RAB associated with the RMS call.
OA88 1669 :
OA88 1670 : IMPLICIT INPUTS:
OA88 1671 : NONE
OA88 1672 :
OA88 1673 : OUTPUT PARAMETERS:
OA88 1674 : NONE
OA88 1675 :
OA88 1676 : IMPLICIT OUTPUTS:
OA88 1677 : Error message
OA88 1678 :
OA88 1679 : COMPLETION CODES:
OA88 1680 : NONE
OA88 1681 :
OA88 1682 : SIDE EFFECTS:
OA88 1683 : Program may exit, depending on severity of the error.
OA88 1684 :
OA88 1685 :--
OA88 1686
OA88 1687 RMS_ERROR:
OFFC OA88 1688 .WORD ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; Entry mask
OA8A 1689
56 04 AC DO OA8A 1690 MOVL 4(AP),R6 ; See whether we're dealing with...
66 03 91 OA8E 1691 CMPB #FAB$C_BID,FAB$B_BID(R6) ; ...a FAB or a RAB
16 12 OA91 1692 BNEQ 10$ ; BR if it's a RAB
57 01F5'CF DE OA93 1693 MOVAL FILE,R7 ; FAB-specific code: text string...
58 56 DO OA98 1694 MOVL R6,R8 ; ...address of FAB...
0C A6 DD OA9B 1695 PUSHL FAB$STV(R6) ; ...STV field for error...
08 A6 DD OA9E 1696 PUSHL FAB$STS(R6) ; ...STS field for error...
0146'CF 08 A6 DO OAA1 1697 MOVL FAB$STS(R6),STATUS ; ...and save the error code
15 11 OAA7 1698 BRB COMMON ; FAB and RAB share other code
OAA9 1699 10$:
57 0201'CF DE OAA9 1700 MOVAL RECORD,R7 ; RAB-specific code: text string...
58 3C A6 DO OAAE 1701 MOVL RAB$FAB(R6),R8 ; ...address of associated FAB...
0C A6 DD OAB2 1702 PUSHL RAB$STV(R6) ; ...STV field for error...
08 A6 DD OAB5 1703 PUSHL RAB$STS(R6) ; ...STS field for error...
0146'CF 08 A6 DO OAB8 1704 MOVL RAB$STS(R6),STATUS ; ...and save the error code
OABE 1705 COMMON:
5A 34 A8 9A OABE 1706 MOVZBL FAB$B_FNS(R8),R10
OAC2 1707 $FAO_S CTRSTR = RMS_ERR_STRING,- ; Common code, prepare error message...
OAC2 1708 OUTLEN = BUFFER_PTR,-
OAC2 1709 OUTBUF = FAO_BUF,-
OAC2 1710 P1 = R7,-
OAC2 1711 P2 = R10,-
OAC2 1712 P3 = FAB$L_FNA(R8)
000C'CF DF OADC 1713 PUSHAL BUFFER_PTR ; ...and arguments for ERROR_EXIT...
```



```
000F0001 8F DD OAE0 1714 PUSHL #^XF0001 ; ...
00741130 8F DD OAE6 1715 PUSHL #UETP$ TEXT ; ...
00 EF OAE6 1716 EXTZV #STSSV-SEVERITY,-
03 OAE6 1717 #STSS$-SEVERITY,-
59 0146'CF 03 OAEF 1718 STATUS,R9 ; ...get the severity code...
6E 59 88 OAF3 1719 R9,(SP) ; ...and add it into the signal name
55555555 8F 18 A8 D1 OAF6 1720 CMPL FAB$L CTX(R8),#DISK_K_PATTERN ; A test or an overhead file?
05 13 OAFE 1721 BEQL TEST_FILE_ERROR ; BR if it's a test file
05 DD OB00 1722 PUSHL #5 ; Current arg count
0097 31 OB02 1723 BRW ERROR_EXIT ; Can't complete test normally
OB05 1724
OB05 1725 TEST_FILE_ERROR:
5B 0146'CF D4 OB05 1726 CCRL STATUS ; Recoverable error, reset exit status
FEFO C8 DE OB09 1727 MOVAL -UETUNT$K FAB(R8),R11 ; Point to node for this FAB
OE AB 01 B1 OB0E 1728 CMPW #DISK_K_CREATE,UETUNT$W_FUNC(R11) ; Were we doing a $CREATE?
14 13 OB12 1729 BEQL 10$ ; BR if so (all errors count)
OB14 1730
OB14 1731 ; We sometimes expect SSS_EXDISKQUOTA or RMSS_FUL errors as a natural
OB14 1732 ; consequence of trying to extend a test file. Ignore them here and let
OB14 1733 ; inline code where the $EXTEND is done handle the error. That same code will
OB14 1734 ; prevent further attempts at expansion because R0 is immediately returned
OB14 1735 ; with an error value and DISK_M_NOEXTEND is set for the file. Note that the
OB14 1736 ; RET instruction below will clean up the stack.
OC AB 000003EC 8F D1 OB14 1737 CMPL #SS$_EXDISKQUOTA,FAB$L_STV(R8) ; Quota exhausted error?
3A 13 OB1C 1738 BEQL 20$ ; BR if so
OB AB 00000000'8F D1 OB1E 1739 CMPL #RMSS_FUL,FAB$L_STS(R8) ; Error because unit was full?
30 13 OB26 1740 BEQL 20$ ; BR if so
OB28 1741
OB28 1742 10$:
0142'CF D6 OB28 1743 INCL ERROR_COUNT ; Keep a running error count
0142'CF DD OB2C 1744 PUSHL ERROR_COUNT ; Have the error stand out in log file
00A0'CF DF OB30 1745 PUSHAL PROCESS_NAME ; ...
00010002 8F DD OB34 1746 PUSHL #^X10002 ; ...
00748020 8F DD OB3A 1747 PUSHL #UETP$ ERBOXPROC ; Set the message code...
6E 59 88 OB40 1748 BISB2 R9,(SP) ; ...and the severity
00000000'GF 09 FB OB43 1749 CALLS #9,G^LIB$SIGNAL ; Let the user know what happened
02 8A OB4A 1750 BICB2 #UETUNT$M TESTABLE,- ; Indicate that this file is no good
OB AB OB4C 1751 UETUNT$B_FLAGS(R11)
01 OE AB B1 OB4E 1752 CMPW UETUNT$W_FUNC(R11),#DISK_K_CREATE ; Were we doing a $CREATE?
04 12 OB52 1753 BNEQ 20$ ; BR if not
04 88 OB54 1754 BISB2 #DISK_M_NOCREATE,- ; Prevent trying to $CLOSE
OB AB OB56 1755 UETUNT$B_FLAGS(R11)
58 5B D0 OB58 1756 20$:
FD9B 30 OB5B 1757 MOVL R11,R8 ; Set up args to see...
04 04 OB5E 1758 BSBW DESTP_CHECK ; ...if additional message is needed
OB5F 1760 RET ; No more I/O to this file
```



```
0B5F 1762 .SBTTL CTRL/C Handler
0B5F 1763 :++
0B5F 1764 : FUNCTIONAL DESCRIPTION:
0B5F 1765 : This routine handles CTRL/C AST's by printing a message and exiting.
0B5F 1766 :
0B5F 1767 : CALLING SEQUENCE:
0B5F 1768 : Called via AST
0B5F 1769 :
0B5F 1770 : INPUT PARAMETERS:
0B5F 1771 : NONE
0B5F 1772 :
0B5F 1773 : IMPLICIT INPUTS:
0B5F 1774 : NONE
0B5F 1775 :
0B5F 1776 : OUTPUT PARAMETERS:
0B5F 1777 : NONE
0B5F 1778 :
0B5F 1779 : IMPLICIT OUTPUTS:
0B5F 1780 : $EXIT status value
0B5F 1781 :
0B5F 1782 : COMPLETION CODES:
0B5F 1783 : NONE
0B5F 1784 :
0B5F 1785 : SIDE EFFECTS:
0B5F 1786 : NONE
0B5F 1787 :
0B5F 1788 : --
0B5F 1789 :
0B5F 1790 CCASTHAND:
OFFC 0B5F 1791 .WORD ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; Entry mask
0B61 1792
00A3'CF DF 0B61 1793 PUSHAL CNTRLCMSG ; Set message pointer
01 DD 0B65 1794 PUSHL #1 ; Set arg count
00741130 8F DD 0B67 1795 PUSHL #UETP$_TEXT!STSSK_WARNING ; Set signal name
00 DD 0B6D 1796 PUSHL #0 ; Indicate an abnormal termination
00A0'CF DF 0B6F 1797 PUSHAL PROCESS_NAME ; ...
02 DD 0B73 1798 PUSHL #2 ; ...
007410E0 8F DD 0B75 1799 PUSHL #UETP$_ABEND!STSSK_WARNING ; ...
00000000'GF 07 FB 0B7B 1800 CALLS #7,G^LIB$SIGNAL ; Output the message
DO 0B82 1801 MOVL #<STSSM_INHIB_MSG!- ; Set the exit status
0B83 1802 SS$ CONTROLC=-
0B83 1803 STSSK_SUCCESS+STSSK_WARNING>,-
0146'CF 10000650 8F 0B83 1804 STATUS
0002'CF 02 A8 0B8B 1805 BISW2 #TEST_OVERM,FLAG ; Indicate that the test is over...
04 0B90 1806 $WAKE_S ; ...and allow it...
0B9B 1807 RET ; ...to terminate cleanly
```



```
OB9C 1809 .SBTTL Error Exit
OB9C 1810 :++
OB9C 1811 : FUNCTIONAL DESCRIPTION:
OB9C 1812 :   This routine prints an error message and exits.
OB9C 1813 :
OB9C 1814 : CALLING SEQUENCE:
OB9C 1815 :   MOVx error status value,STATUS
OB9C 1816 :   PUSHx error specific information on the stack
OB9C 1817 :   PUSHL current argument count
OB9C 1818 :   BRW ERROR_EXIT
OB9C 1819 :
OB9C 1820 : INPUT PARAMETERS:
OB9C 1821 :   Arguments to LIB$SIGNAL, as above
OB9C 1822 :
OB9C 1823 : IMPLICIT INPUTS:
OB9C 1824 :   NONE
OB9C 1825 :
OB9C 1826 : OUTPUT PARAMETERS:
OB9C 1827 :   Message to SYS$OUTPUT and SYS$ERROR
OB9C 1828 :
OB9C 1829 : IMPLICIT OUTPUTS:
OB9C 1830 :   Program exit
OB9C 1831 :
OB9C 1832 : COMPLETION CODES:
OB9C 1833 :   NONE
OB9C 1834 :
OB9C 1835 : SIDE EFFECTS:
OB9C 1836 :   NONE
OB9C 1837 :
OB9C 1838 :--
OB9C 1839 :
OB9C 1840 ERROR_EXIT:
OB9C 1841
OB9C 1842 $SETAST_S ENBFLG = #0 ; ASTs can play havoc with messages
15 0002'CF 03 EO OBA5 1843 BBS #BEGIN_MSGV,FLAG,10$ ; BR if 'begin' msg already printed
OB9C 1844 CLRL -(SP) ; Set the time stamp flag
OB9C 1845 PUSHAL TEST_NAME ; Set the test name
OB9C 1846 PUSHL #2 ; Push the argument count
OB9C 1847 PUSHL #UETP$_BEGINDD!ST$K_SUCCESS ; Set the message code
00000000'GF 04 FB OBB9 1848 CALLS #4,G^LIB$SIGNAL ; Print the startup message
OB9C 1849 10$:
OB9C 1850 ADDL3 (SP)+, #8, ARG_COUNT ; Get total # args, pop partial count
OB9C 1851 INCL ERROR_COUNT ; Keep running error count
OB9C 1852 PUSHL #0 ; Push the time parameter
OB9C 1853 PUSHAL PROCESS_NAME ; Push test name...
OB9C 1854 PUSHL #^XF0002 ; ...arg count...
OB9C 1855 PUSHL #UETP$_ABENDDD!ST$K_ERROR ; ...and signal name
OB9C 1856 PUSHL ERROR_COUNT ; Finish off arg list...
OB9C 1857 PUSHAL PROCESS_NAME ; ...
OB9C 1858 PUSHL #^X10002 ; ...
OB9C 1859 PUSHL #UETP$_ERBOXPROC!ST$K_ERROR ; ...for error box message
00000000'GF 018A'CF FB OBF0 1860 CALLS ARG_COUNT,G^LIB$SIGNAL ; Truly bitch
OB9C 1861
OB9C 1862 TSTL STATUS ; Did we exit with an error code?
OB9C 1863 BNEQ 20$ ; BR if we did
007410E2 8F DO OBF9 1864 MOVL #UETP$_ABENDDD!ST$K_ERROR,- ; Supply a generic one otherwise
OB9C 1865 OC05 1865 STATUS
```


UETDISK00
V04-000

- VAX/VMS UETP DISK EXERCISER
Error Exit

C 16

16-SEP-1984 01:22:47 VAX/VMS Macro V04-00
5-SEP-1984 04:24:57 [UETP.SRC]UETDISK00.MAR;1

Page 45
(29)

0146'CF 10000000 8F CB 0C08 1866 20\$:
0C08 1867
0C11 1868

BISL #STSSM-INHIB_MSG,STATUS ; Don't print messages twice!
\$EXIT_S STATUS ; Exit in error


```

OC1C 1870      .SBTTL Exit Handler
OC1C 1871      :++
OC1C 1872      : FUNCTIONAL DESCRIPTION:
OC1C 1873      :   This routine handles cleanup at exit.  If the MODE logical name is
OC1C 1874      :   equated to 'ONE', the routine will update the test flag in the
OC1C 1875      :   UETINIDEV.DAT file depending on the UETUNTSM_TESTABLE flag state in the
OC1C 1876      :   UETUNTSB_FLAGS field of the unit blocks corresponding to a line in the
OC1C 1877      :   file.
OC1C 1878      :
OC1C 1879      : CALLING SEQUENCE:
OC1C 1880      :   Invoked automatically by $EXIT System Service.
OC1C 1881      :
OC1C 1882      : INPUT PARAMETERS:
OC1C 1883      :   STATUS contains the exit status.
OC1C 1884      :   FLAG has synchronizing bits.
OC1C 1885      :   DDB_RFA contains the RFA of the DDB record for this device in UETINIDEV
OC1C 1886      :
OC1C 1887      : IMPLICIT INPUTS:
OC1C 1888      :   UNIT_LIST points to the head of a doubly linked circular list of unit
OC1C 1889      :   blocks for the device under test.
OC1C 1890      :
OC1C 1891      : OUTPUT PARAMETERS:
OC1C 1892      :   NONE
OC1C 1893      :
OC1C 1894      : IMPLICIT OUTPUTS:
OC1C 1895      :   Various files are de-accessed, the process name is reset, and any
OC1C 1896      :   necessary synchronization with UETPDEV01 is carried out.
OC1C 1897      :   If the MODE logical name is equated to 'ONE', the routine will update
OC1C 1898      :   the test flag in the UETINIDEV.DAT file depending on the
OC1C 1899      :   UETUNTSM_TESTABLE flag state in the UETUNTSB_FLAGS field of the unit
OC1C 1900      :   blocks corresponding to the units on the disk controller.
OC1C 1901      :
OC1C 1902      : COMPLETION CODES:
OC1C 1903      :   NONE
OC1C 1904      :
OC1C 1905      : SIDE EFFECTS:
OC1C 1906      :   NONE
OC1C 1907      :
OC1C 1908      : --
OC1C 1909      :
OC1C 1910      : EXIT_HANDLER:
OFFC OC1C 1911      .WORD  ^M<R2,R3,R4,R5,R6,R7,R8,R9,R10,R11> ; Entry mask
OC1E 1912
OC1E 1913      $SETSM_S ENBFLG = #0 ; Turn off System Service failure mode
OC27 1914      $SETAST_S ENBFLG = #0 ; We're finished - no more ASTs
03 0002'CF 04 E0 OC30 1915      BBS  #ONE_SHOTV,FLAG,10$ ; If one-shot, update testability...
00CD 31 OC36 1916      BRW  END_UPDATE ; ...else don't update UETINIDEV.DAT
OC39 1917 10$:
03 0002'CF 02 E0 OC39 1918      BBS  #SAFE_TO_UPDV,FLAG,20$ ; Only update if it's safe
00C4 31 OC3F 1919      BRW  END_UPDATE ; Else forget it
OC42 1920 20$:
5A 0A88'CF DE OC42 1921      MOVAL INI RAB,R10 ; Set the RAB address
1E AA 02 90 OC47 1922      MOVB  #RAB$C_RFA,RAB$B_RAC(R10) ; Set RFA mode
10 AA 0ACC'CF 06 28 OC4B 1923      MOVBC3 #6,DDB_RFA,RAB$W_RFA(R10) ; Set RFA to DDB line
OC52 1924      $GET RAB = (R10) ; Go back to the DDB record
03 50 E8 OC5B 1925      BLBS  R0,30$
0087 31 OC5E 1926      BRW  UPDATE_FAILED ; If failure then forget it
```



```
5B 0190'CF 1E AA 00 90 0C61 1927 30$:
00000190'8F C1 0C61 1928 MOV#RAB$C_SEQ,RAB$B_RAC(R10) ; Set back to sequential mode
59 D4 0C65 1929 ADDL3 #UNIT_CIST,UNIT_CIST,R11 ; Set the unit block list header
01 E1 0C6F 1930 CLRL R9 ; Init a counter
02 0B AB 01 0C71 1931 UNIT_LOOP:
59 D6 0C71 1932 BBC #UETUNT$V_TESTABLE,- ; BR if this unit is not testable
01 0C73 1933 UETUNT$B_FLAGS(R11),10$
59 D6 0C76 1934 INCL R9 ; Count testable units
5B 6B C0 0C78 1935 10$:
00000190'8F 5B D1 0C78 1936 ADDL2 (R11),R11 ; Next unit block
ED 12 0C7B 1937 CMPL R11,#UNIT_LIST ; Are we full circle in the list?
59 D5 0C82 1938 BNEQ UNIT_LOOP ; BR if not
12 12 0C84 1939 TSTL R9 ; Any testable units?
0018'CF 4E 8F 90 0C86 1940 BNEQ 20$ ; BR if yes...
4E 50 E9 0C88 1941 MOV#^A/N/,BUFFER+4 ; ...else disable the DDB record...
0C8E 1942 $UPDATE RAB = (R10) ; ...here
0C97 1943 BLBC RO,UPDATE_FAILED ; If error then forget it
0C9A 1944 20$:
5B 6B C0 0C9A 1945 ADDL2 (R11),R11 ; Next unit block
00000190'8F 5B D1 0C9D 1946 CMPL R11,#UNIT_LIST ; Are we full circle in the list?
60 13 0CA4 1947 BEQL END_UPDATE ; BR if yes
01A6 CB 95 0CA6 1948 TSTB DISK_B_FILE(R11) ; Is this node for file 0 on a unit?
EE 12 0CAA 1949 BNEQ 20$ ; BR if not; we'll only $GET for first node
30 50 E9 0CAC 1950 $GET RAB = (R10) ; Get a record
0014'CF 20 8A 0CB5 1951 BLBC RO,UPDATE_FAILED ; If error then forget it
0014'CF 55 8F 91 0CB8 1952 BICB2 #LC_BITM,BUFFER ; Convert to uppercase
41 12 0CC3 1953 CMPB #^A7U/,BUFFER ; Is it a UCB record?
01 E0 0CC5 1954 BNEQ END_UPDATE ; BR if not
5B 0B AB 01 0CC7 1955 BBS #UETUNT$V_TESTABLE,- ; BR if this unit is testable
01A7 CB D0 0CCA 1956 UETUNT$B_FLAGS(R11),20$
05 13 0CCF 1957 MOVL DISK_L_OTHER_PTR(R11),R8 ; See if other file on this unit...
02 E0 0CD1 1958 BEQL 30$ ; ...if there is one...
C4 0B A8 0CD3 1959 BBS #UETUNT$M_TESTABLE,- ; ...might be testable
0018'CF 4E 8F 90 0CD6 1960 30$:
B2 50 E8 0CD6 1961 MOV#^A/N/,BUFFER+4 ; None testable, disable UCB record...
0C AA DD 0CDC 1962 $UPDATE RAB = (R10) ; ...here
50 DD 0CE5 1963 BLBS RO,20$ ; Look at the next record if no error
01B8'CF DF 0CE8 1964 UPDATE_FAILED:
01 DD 0CE8 1965 PUSH#RAB$L_STV(R10) ; Do a simple message...
00 EF 0CEB 1966 PUSH#RO ; ...to tell of the failure
7E 50 03 0CED 1967 PUSH#INIDEV_UPDERR
6E 00741130 8F C8 0CF1 1968 PUSH#1
00000000'GF 05 FB 0CF3 1969 EXTZV #ST$V_SEVERITY,- ; Copy the severity from RMS status...
03 0002'CF 01 E0 0CF5 1970 #ST$S_SEVERITY,RO,-(SP) ; ...to our message
FB58 30 0CF8 1971 #UETP$-TEXT,(SP)
00 DD 0CF8 1972 CALLS #5,G^LIB$SIGNAL
02 DD 0CF9 1973 END_UPDATE:
00 EF 0D06 1974 BBS #TEST_OVERV,FLAG,10$ ; Did the test end normally (timeout)?
7E 0146'CF 01 E0 0D0C 1975 RSBW CLEAN_UP ; It didn't so must erase files
00 DD 0D0F 1976 10$:
000F'CF DF 0D0F 1977 PUSH#0 ; Set the time flag
02 DD 0D11 1978 PUSH#TEST_NAME ; Push the test name
00 DD 0D15 1979 PUSH#2 ; Push arg count
03 EF 0D17 1980 EXTZV #ST$V_SEVERITY,- ; Push the proper exit severity...
01A 1981 #ST$S_SEVERITY,-
1982 STATUS,-(SP)
1983
```


UETDISK00
V04-000

- VAX/VMS UETP DISK EXERCISER
Exit Handler

F 16

16-SEP-1984 01:22:47 VAX/VMS Macro V04-00
5-SEP-1984 04:24:57 [UETP.SRC]UETDISK00.MAR;1

Page 48
(30)

6E	00741080	8F	C8	0D1E	1984	BISL2	#UETP\$_ENDEDD,(SP)	; ...and use it in our message code
		04	DD	0D25	1985	PUSHL	#4	
	51	5E	D0	0D27	1986	MOVL	SP,R1	
				0D2A	1987	\$PUTMSG_S	MSGVEC = (R1)	; Output the message
				0D39	1988	\$SETPRN_S	PRCNAM = ACNT_NAME	; Reset the process name
			04	0D44	1989	RET		; That's all folks!
				0D45	1990			
				0D45	1991			
						.END	UETDISK00	

UETDISK00
Symbol table

- VAX/VMS UETP DISK EXERCISER

G 16

16-SEP-1984 01:22:47 VAX/VMS Macro V04-00
5-SEP-1984 04:24:57 [UETP.SRC]UETDISK00.MAR;1

Page 49
(30)

\$\$.TAB	= 00000BB8	R	04	DISK_V_NOCREATE	= 00000002		
\$\$.TABEND	= 00000C18	R	04	DISK_V_NOEXTEND	= 00000003		
\$\$.TMP	= 00000000			DISK_W_UNIT	000001A4		
\$\$.TMP1	= 00000001			DO THE WRITE	00000262	R	03
\$\$.TMP2	= 0000006A			DUMMY_FAB	00000785	R	06
\$\$.TMPX	= 00000016	R	05	DUMMY_NAM	00000B24	R	04
\$\$.TMPX1	= 0000000D			DUMMY_RAB	00000BB8	R	04
\$\$ T1	= 00000000			DVIS_DEVNAM	00000B74	R	04
\$\$ T2	= 00000006			EFN2	= 00000020		
ACNT_NAME	00000000	R	03	END UPDATE	= 00000004		
ALL SET	00000414	R	06	ERROR_COUNT	00000D06	R	06
ARG_COUNT	0000018A	R	04	ERROR_EXIT	00000142	R	04
BAKERS	= 00000014			EXIT_DESC	00000B9C	R	06
BEGIN_MSGM	= 00000008			EXIT_HANDLER	0000017A	R	04
BEGIN_MSGV	= 00000003			FAB\$B_BID	00000C1C	R	06
BUFFER	00000014	R	04	FAB\$B_BLN	= 00000000		
BUFFER_PTR	0000000C	R	04	FAB\$B_DNS	= 00000001		
CCASTHAND	00000B5F	R	06	FAB\$B_FNS	= 00000035		
CHECK_EXTEND	00000735	R	06	FAB\$C_BID	= 00000034		
CHFSL_SIGARGLST	= 00000004			FAB\$C_BLN	= 00000003		
CHFSL_SIG_ARG1	= 00000008			FAB\$C_SEQ	= 00000050		
CHFSL_SIG_ARGS	= 00000000			FAB\$C_UDF	= 00000000		
CHFSL_SIG_NAME	= 00000004			FAB\$C_VAR	= 00000002		
CLEAN_UP	00000867	R	06	FAB\$L_ALQ	= 00000010		
CNTRLMSG	000000A3	R	03	FAB\$L_CTX	= 00000018		
COMMON	00000ABE	R	06	FAB\$L_DEV	= 00000040		
CONTROLLER	00000031	R	03	FAB\$L_DNA	= 00000030		
CONT_DESC	000001ED	R	03	FAB\$L_FNA	= 0000002C		
CREATE_CHECK	00000935	R	06	FAB\$L_FOP	= 00000004		
CS1	00000082	R	03	FAB\$L_NAM	= 00000028		
CS3	00000094	R	03	FAB\$L_STS	= 00000008		
DDB_RFA	00000ACC	R	04	FAB\$L_STV	= 0000000C		
DEAD_CTRLNAME	000000E4	R	03	FAB\$V_BIO	= 00000005		
DESTP_CHECK	000008F9	R	06	FAB\$V_CHAN_MODE	= 00000002		
DEV\$V_TRM	= 00000002			FAB\$V_CR	= 00000001		
DEVDEP_SIZE	= 00000176			FAB\$V_FILE_MODE	= 00000004		
DEVDS	00000098	R	04	FAB\$V_GET	= 00000001		
DEVNAM_LEN	00000164	R	04	FAB\$V_LNM_MODE	= 00000000		
DEV_NAME	000000B7	R	04	FAB\$V_PUT	= 00000000		
DIB	000000C6	R	04	FAB\$V_UFO	= 00000011		
DIB\$B_DEVCLASS	= 00000004			FAB\$V_UPD	= 00000003		
DIB\$B_DEVTYPE	= 00000005			FAB\$V_UPI	= 00000006		
DIB\$K_LENGTH	= 00000074			FAB\$W_GBC	= 00000048		
DIB\$L_MAXBLOCK	= 00000070			FAO_BUF	00000004	R	04
DIBBUF	000000CE	R	04	FILE	000001F5	R	03
DISK_B_FILE	000001A6			FIND_IT	00000244	R	06
DISK_C_NAM	000001BB			FLAG	00000002	R	04
DISK_K_BUFFER	0000031C			FN_LEN	= 00000027		
DISK_K_CREATE	= 00000001			FOUND_IT	000002DC	R	06
DISK_K_PATTERN	= 55555555			GET_PATTERN	000007E5	R	06
DISK_L_OTHER_PTR	000001A7			ILLEGAL_REC	00000151	R	03
DISK_L_SIZE	000001AF			INADDRESS	00000152	R	04
DISK_L_TOP_PCT	000001AB			INIDEV_UPDERR	000001B8	R	03
DISK_M_NOCREATE	= 00000004			INI_FAB	00000A38	R	04
DISK_M_NOEXTEND	= 00000008			INI_RAB	00000A88	R	04
DISK_Q_FILEDESC	000001B3			INPUT_ITMLST	00000072	R	03
DISK_T_NAMFILSPC	0000021B						

UETDISK00
Symbol table

- VAX/VMS UETP DISK EXERCISER

H 16

16-SEP-1984 01:22:47 VAX/VMS Macro V04-00
5-SEP-1984 04:24:57 [UETP.SRC]UETDISK00.MAR;1

Page 50
(30)

IOSM_CTRLCAST	*****	X	06	RABSL_ROP	= 00000004		
IOS_SETMODE	*****	X	06	RABSL_STS	= 00000008		
ITERATION	0000016E	R	04	RABSL_STV	= 0000000C		
LC_BITM	= 00000020			RABSL_UBF	= 00000024		
LIBSSIGNAL	*****	X	06	RABSM_ASY	= 00000001		
MAX_DEV_DESIG	= 0000000A			RABSV_ASY	= 00000000		
MAX_PROC_NAME	= 0000000F			RABSV_BIO	= 0000000B		
MAX_UNIT_DESIG	= 00000005			RABSV_PMT	= 0000001E		
MODE	00000041	R	03	RABSW_RFA	= 00000010		
MSG_BLOCK	00000176	R	04	RABSW_RSZ	= 00000022		
NAMSB_ESS	= 0000000A			RANDOM1	00000166	R	04
NAMSB_NOP	= 00000008			RANDOM2	0000016A	R	04
NAMSB_RSL	= 00000003			RANDOM_DATA	000001A0	R	04
NAMSB_RSS	= 00000002			READ_NEXT	000007FA	R	06
NAMSC_BID	= 00000002			READ_SIZE	= 00000800		
NAMSC_BLN	= 00000060			RECORD	00000201	R	03
NAMSC_MAXRSS	= 000000FF			REC_SIZE	= 00000028		
NAMSL_ESA	= 0000000C			RESTART	00000455	R	06
NAMSL_RSA	= 00000004			RMSS_BLN	*****	X	03
NAME_CEN	= 0000000F			RMSS_BUSY	*****	X	03
NEW_NODE	00000198	R	04	RMSS_CDA	*****	X	03
NOUNIT_SELECTED	0000012B	R	03	RMSS_DNF	*****	X	06
NO_CTRLNAME	000000C4	R	03	RMSS_FAB	*****	X	03
NO_RMS_AST_TABLE	0000004D	R	03	RMSS_FACILITY	= 00000001		
NRAT_LENGTH	= 00000014			RMSS_FUL	*****	X	06
ONE_SHOTM	= 00000010			RMSS_RAB	*****	X	03
ONE_SHOTV	= 00000004			RMS_ERROR	00000A88	R	06
ONE_SHOT_TEST	0000051E	R	06	RMS_ERR_STRING	0000020F	R	03
OTSSCVT_T1_L	*****	X	06	SAFE_TO_UPDM	= 00000004		
OTSSCVT_T2_L	*****	X	06	SAFE_TO_UPDV	= 00000002		
OUTADDRESS	0000015A	R	04	SECSM_EXPREG	*****	X	06
PAGES	= 00000006			SECSM_GBL	*****	X	06
PASS	00000172	R	04	SHRS_ABENDD	= 000010E0		
PASS_MSG	00000185	R	03	SHRS_BEGINDD	= 00001038		
PATTERN	00000267	R	03	SHRS_ENDEDD	= 00001080		
PATTERNM	= 00000020			SHRS_OPENIN	= 00001098		
PATTERNV	= 00000005			SHRS_TEXT	= 00001130		
PATTERN_DATA	000009A0	R	04	SSS_BADPARAM	= 00000014		
PERCENTAGE	= CCCC3E4C			SSS_CONTROLC	= 00000651		
PMTSIZ	= 00000019			SSS_EXDISKQUOTA	= 000003EC		
PRE_CLEAN	00000827	R	06	SSS_NORMAL	= 00000001		
PROCESS_NAME	000000A0	R	04	SSS_NOSUCHSEC	= 00000978		
PROCESS_NAME_FREE	= 0000000B			SSS_SSFAIL	= 0000045C		
PROC_CONT_NAME	0000008B	R	06	SSS_WASSET	= 00000009		
PROMPT	00000230	R	03	SSERROR	000009A5	R	06
QUAD_STATUS	0000014A	R	04	SS_SYNCH_EFN	= 00000003		
RABSB_PSZ	= 00000034			STATUS	00000146	R	04
RABSB_RAC	= 0000001E			STRUPCASE	*****	X	06
RABSC_BID	= 00000001			STSSK_ERROR	= 00000002		
RABSC_BLN	= 00000044			STSSK_INFO	= 00000003		
RABSC_RFA	= 00000002			STSSK_SUCCESS	= 00000001		
RABSC_SEQ	= 00000000			STSSK_WARNING	= 00000000		
RABSL_BKT	= 00000038			STSSM_INHIB_MSG	= 10000000		
RABSL_CTX	= 00000018			STSSS_FAC_NO	= 0000000C		
RABSL_FAB	= 0000003C			STSSS_SEVERITY	= 00000003		
RABSL_PBF	= 00000030			STSSV_FAC_NO	= 00000010		
RABSL_RBF	= 00000028			STSSV_SEVERITY	= 00000000		

UETDISK00
Symbol table

- VAX/VMS UETP DISK EXERCISER

I 16

16-SEP-1984 01:22:47 VAX/VMS Macro V04-00
5-SEP-1984 04:24:57 [UETP.SRC]UETDISK00.MAR;1

Page 51
(30)

SUC_EXIT	0000058B	R	06
SUPDEV_GBLSEC	00000020	R	03
SUP_FAB	00000AD4	R	04
SYSS\$ASSIGN	*****	GX	06
SYSS\$CLOSE	*****	GX	06
SYSS\$CONNECT	*****	GX	06
SYSS\$CREATE	*****	GX	06
SYSS\$CRMPSC	*****	GX	06
SYSS\$DCLEXH	*****	GX	06
SYSS\$ERASE	*****	GX	06
SYSS\$EXIT	*****	GX	06
SYSS\$EXPREG	*****	GX	06
SYSS\$EXTEND	*****	GX	06
SYSS\$FAO	*****	X	06
SYSS\$GET	*****	GX	06
SYSS\$GETDEV	*****	GX	06
SYSS\$GETDVI	*****	GX	06
SYSS\$GETMSG	*****	GX	06
SYSS\$HIBER	*****	GX	06
SYSS\$INPUT	00000061	R	03
SYSS\$MGBLSC	*****	GX	06
SYSS\$OPEN	*****	GX	06
SYSS\$PUTMSG	*****	GX	06
SYSS\$QIOW	*****	GX	06
SYSS\$READ	*****	GX	06
SYSS\$SETAST	*****	GX	06
SYSS\$SETIMR	*****	GX	06
SYSS\$SETPRN	*****	GX	06
SYSS\$SETSPM	*****	GX	06
SYSS\$TRNLOG	*****	GX	06
SYSS\$UPDATE	*****	GX	06
SYSS\$WAIT	*****	GX	06
SYSS\$WAKE	*****	GX	06
SYSS\$WRITE	*****	GX	06
SYS0_SYSTEST_DIR	00000253	R	03
SYSIN_FAB	000009A4	R	04
SYSIN_RAB	000009F4	R	04
SYSTEST_DIR	00000249	R	03
TEST_FICE_ERROR	00000B05	R	06
TEST_NAME	0000000F	R	03
TEST_OVER	000005F7	R	06
TEST_OVERM	= 00000002		
TEST_OVERV	= 00000001		
TEST_OVER STATUS	00000600	R	06
TEXT_BUFFER	= 00000084		
THREEMIN	000001DD	R	03
TIME_IT	00000442	R	06
TIME_OUT	00000992	R	06
TOP_PCT	= 00004040		
TTCRAN	00000000	R	04
UETDISK00	00000000	RG	06
UETP	= 00740000		
UETPS_ABEND	= 007410E0		
UETPS_ABORTC	= 0074832B		
UETPS_BEGIN	= 00741038		
UETPS_DATADEVERR	= 00748018		
UETPS_DENOSU	= 00748333		

UETPS_DESTP	= 007481A2		
UETPS_ENDEDD	= 00741080		
UETPS_ERBOXPROC	= 00748020		
UETPS_FACILITY	= 00000074		
UETPS_OPENIN	= 00741098		
UETPS_TEXT	= 00741130		
UETUNTSB_FLAGS	= 0000000B		
UETUNTSB_TYPE	= 00000008		
UETUNTSC_DEVDEP	= 000001A4		
UETUNTSC_FAB	= 00000110		
UETUNTSC_INDSIZ	= 000001A4		
UETUNTSK_FAB	= 00000110		
UETUNTSK_RAB	= 00000160		
UETUNTSK_ITER	= 00000010		
UETUNTSM_TESTABLE	= 00000002		
UETUNTST_FILSPC	= 00000014		
UETUNT\$V_TESTABLE	= 00000001		
UETUNT\$W_FUNC	= 0000000E		
UETUNT\$W_SIZE	= 00000009		
UNIT_DESC	000001E5	R	03
UNIT_FILE_SETUP	0000060B	R	06
UNIT_LIST	00000190	R	04
UNIT_LOOP	00000C71	R	06
UNIT_NUMBER	00000162	R	04
UPDATE_FAILED	00000CE8	R	06
VERIFY_ERROR	000008A8	R	06
WRITE_NEXT	000006F0	R	06
WRITE_SIZE	= 00000800		

+-----+
! Psect synopsis !
+-----+

PSECT name	Allocation	PSECT No.	Attributes
ABS	00000000 (0.)	00 (0.)	NOPIC USR CON ABS LCL NOSHR NOEXE NORD NOWRT NOVEC BYTE
\$ABSS	00000000 (0.)	01 (1.)	NOPIC USR CON ABS LCL NOSHR EXE RD WRT NOVEC BYTE
DEVDEP_STR_DEF	00000B1C (2844.)	02 (2.)	NOPIC USR CON ABS LCL NOSHR NOEXE RD NOWRT NOVEC PAGE
RODATA	00000276 (630.)	03 (3.)	NOPIC USR CON REL LCL NOSHR NOEXE RD NOWRT NOVEC PAGE
RWDATA	00000C18 (3096.)	04 (4.)	NOPIC USR CON REL LCL NOSHR NOEXE RD WRT NOVEC PAGE
\$RMSNAM	00000023 (35.)	05 (5.)	NOPIC USR CON REL LCL NOSHR EXE RD WRT NOVEC BYTE
DISK	00000D45 (3397.)	06 (6.)	NOPIC USR CON REL LCL NOSHR EXE RD NOWRT NOVEC PAGE

+-----+
! Performance indicators !
+-----+

Phase	Page faults	CPU Time	Elapsed Time
Initialization	27	00:00:00.12	00:00:00.58
Command processing	145	00:00:00.81	00:00:04.09
Pass 1	501	00:00:23.28	00:00:51.25
Symbol table sort	0	00:00:02.25	00:00:04.58
Pass 2	353	00:00:06.20	00:00:11.75
Symbol table output	36	00:00:00.26	00:00:00.47
Psect synopsis output	3	00:00:00.03	00:00:00.04
Cross-reference output	0	00:00:00.00	00:00:00.00
Assembler run totals	1065	00:00:32.97	00:01:12.76

The working set limit was 2000 pages.
126800 bytes (248 pages) of virtual memory were used to buffer the intermediate code.
There were 80 pages of symbol table space allocated to hold 1518 non-local and 67 local symbols.
1991 source lines were read in Pass 1, producing 38 object records in Pass 2.
65 pages of virtual memory were used to define 58 macros.

+-----+
! Macro library statistics !
+-----+

Macro library name	Macros defined
_\$255\$DUA28:[UETP.OBJ]UETP.MLB;1	2
_\$255\$DUA28:[SYS.OBJ]LIB.MLB;1	0
_\$255\$DUA28:[SYSLIB]STARLET.MLB;2	53
TOTALS (all libraries)	55

1859 GETS were required to define 55 macros.

There were no errors, warnings or information messages.

MACRO/LIS=LISS:UETDISK00/OBJ=OBJ\$:UETDISK00 MSRC\$:UETDISK00/UPDATE=(ENH\$:UETDISK00)+EXECML\$/LIB+LIB\$:UETP/LIB

0410 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY